VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Návrh a implementace aplikace pro systém SAP HANA s využitím paralelního zpracování dat

Design and Implementation of Application for the SAP HANA System Using Parallel Data Processing

Student: Bc. Jan Hošna
Vedoucí diplomové práce: doc. RNDr. Ivo Martiník, Ph.D.

Ostrava 2020

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání diplomové práce

Student: **Bc. Jan Hošna**

Studijní program: N6209 Systémové inženýrství a informatika

Studijní obor: 6209T017 Informatika v ekonomice

Téma: Návrh a implementace aplikace pro systém SAP HANA s využitím paralelního zpracování dat
Design and Implementation of Application for the SAP HANA System Using Parallel Data Processing

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Úvod
2. Teoreticko-metodická východiska k vývoji aplikací pro systém SAP HANA
3. Analýza současného stavu
4. Návrh a implementace aplikace k zobrazení rozpadu materiálů
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků diplomové práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

BANDARI, Kiran. *Complete ABAP*. Boston, MA: Rheinwerk Publishing, 2016. 1047 p. ISBN 978-1-4932-1273-6.
GAHM, Hermann et al. *ABAP development for SAP HANA*. 2nd ed. Boston, MA: Rheinwerk Publishing, 2016. 641 p. ISBN 978-1-4932-1305-4.
HARDY, Paul. *ABAP to the future*. 3rd ed. Boston, MA: Rheinwerk Publishing, 2019. 864 p. ISBN 978-1-4932-1761-8.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. RNDr. Ivo Martiník, Ph.D.**

Datum zadání: 22.11.2019

Datum odevzdání: 24.04.2020

Ing. Petr Rozehnal, Ph.D.
vedoucí katedry

doc. Ing. Lenka Kauerová, CSc.
proděkanka pro studium
na základě pověření k jednání č.j.
VSB/19/050319/9900 ze dne 24. 9. 2019

Prohlašuji, že jsem celou diplomovou práci, včetně všech příloh, vypracoval samostatně.

V Ostravě dne 24.04.2020 ..... Hošna .....

Bc. Jan Hošna

Tímto bych chtěl poděkovat vedoucímu mé diplomové práce doc. RNDr. Ivo Martiníkovi, Ph.D. za odborné vedení, cenné rady a připomínky k práci.

Obsah

1 Úvod

Oblast Enterprise Resource Planning (ERP) systémů (Bandari, 2016) je již mnoho let velmi aktuálním tématem, jelikož si většina firem uvědomuje výhody ERP systémů oproti non-ERP systémům. V oblasti informačních technologií postupuje vývoj neustále kupředu, a proto i ERP systémy musí udržovat svou konkurenceschopnost nejen oproti non-ERP systémům, které se snaží své nedostatky minimalizovat, ale i v konkurenci mezi jinými alternativami ERP systémů.

Společnost SAP se neustále snaží o inovaci svého ERP systému s cílem zjednodušení tvorby uživatelských rozhraní a zpracování dat přímo v operační paměti především prostřednictvím platformy SAP HANA, která je nedílnou součástí nového ERP systému SAP s názvem S/4 HANA.

Již jen z tohoto důvodu by bylo vhodné reimplementovat veškeré programy, které nejsou optimalizovány pro databázový systém, který je součástí platformy SAP HANA, jelikož by tím minimálně došlo k zvýšení rychlosti běhu programů.

Dalším důvodem pro optimalizaci stávajících programů na platformě SAP je fakt, že SAP systémy běží ve velkém množství podniků již desítky let. Díky tomu je v těchto systémech mnoho programů, které jsou zastaralé a které obsahují dnes již nedoporučovaná klíčová slova programovacího jazyka ABAP. Tento programovací jazyk se za dobu své existence posunul technologicky mnohem dál, než bylo při jeho zrodu zamýšleno, kdy měl být využíván pouze jako programovací jazyk pro generování reportů.

Společnost SAP má aktuálně největší podíl na trhu ERP systémů oproti ostatním firmám v této oblasti. S dalekosáhlou historií a roky prověřenými v praxi se jeví jako kvalitní volba pro budoucí specializaci technicky vzdělaných vysokoškoláků.

Oblast platformy SAP je velmi rozsáhlá, a proto je nutné, aby firmy měly ve svých ERP týmech jak experty s mnohaletou praxí, tak i juniory snažící se o inovace a podporu nejnovějších přístupů v této oblasti.

Hlavním a prvním cílem diplomové práce je analýza, návrh, implementace a optimalizace programu v programovacím jazyce ABAP pro ERP platformu SAP HANA, která při svém provádění významně využívá paralelního přístupu ke zpracování dat. Program bude využíván k zobrazení rozpadu materiálů na základě materiálového čísla, případně na základě čísla objednávky a pořadí materiálu v objednávce.

Druhým cílem této práce je popis teoretických pojmů a metodik, které souvisejí s vývojem a paralelizací programů v jazyce ABAP pro platformu SAP HANA.

Jejím třetím cílem je pak stručné představení společnosti Siemens s. r. o., kde tato práce vznikala a detailní popis problému, který je v rámci této práce řešen.

Ve druhé kapitole jsou vysvětleny základní pojmy a metodická východiska nutná k vývoji programů a paralelnímu zpracování dat při tvorbě programů v programovacím jazyce ABAP pro platformu SAP HANA, ať se již jedná o historii, verze systémů a jejich vlastnosti, architekturu, jednotlivé moduly v systému SAP, základní typy aplikací v systému SAP, programovací jazyk ABAP, platformu SAP HANA, paralelní zpracování dat, návrhové vzory a metodiky vývoje softwaru.

V kapitole třetí je představena společnost Siemens s. r. o. a současná situace problému, který je v rámci této práce řešen.

Ve čtvrté kapitole je provedena volba metodiky vývoje softwaru a v rámci jednotlivých sprintů jsou řešeny uživatelské požadavky od jejich základních funkcionalit, testování a ošetřování chybových stavů, přes tvorbu cizojazyčných překladů a optimalizaci běhu programu až po návrh a implementaci paralelního zpracování dat a následné shrnutí dosažených výstupů.

V závěru je zhodnoceno využití paralelního zpracování dat pro programy v jazyce ABAP pro platformu SAP HANA a vyhodnocení dosažených výstupů práce oproti cílům, které byly stanoveny.

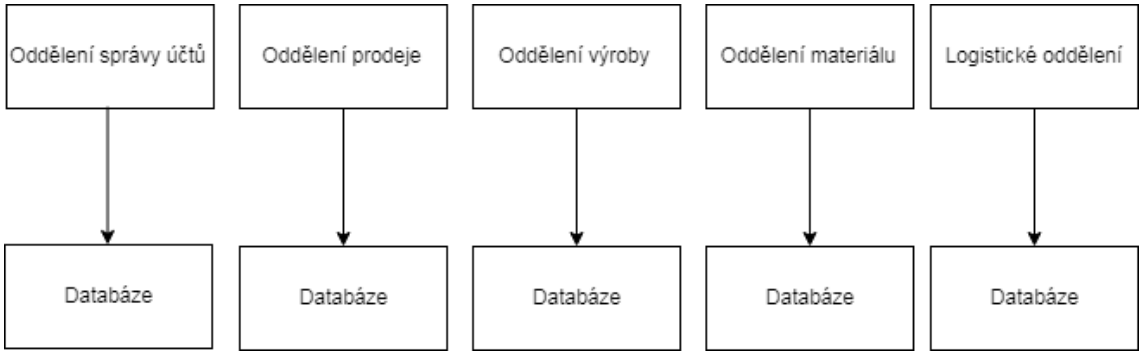
2 Teoreticko-metodická východiska k vývoji aplikací pro systém SAP HANA

Enterprise Resource Planning (ERP – v překladu plánování podnikových zdrojů, příp. podnikový informační systém) je označení programového systému, pomocí kterého daná organizace řídí a integruje všechny nebo většinu svých činností, mezi něž patří např. oblasti účetnictví, nákupu, prodeje, projektového řízení rizik, lidských zdrojů, výroby, logistiky, správy materiálu. (www.tutorialspoint.com)

2.1 Výhody ERP systémů oproti non-ERP systémům

V non-ERP systému má každé oddělení v organizaci svou vlastní databázi a aplikace, což je graficky znázorněno na obrázku 2.1.

Obrázek 2.1 Databáze v non-ERP systému

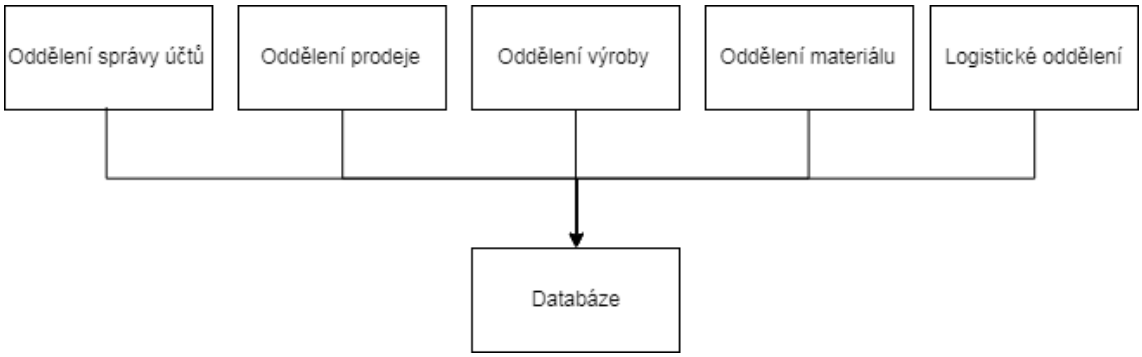


Zdroj: (Bandari, 2016) – vlastní zpracování

V non-ERP systémech nedochází k výměně dat mezi jednotlivými databázemi v reálném čase. Pokud by například došlo k získání nové zakázky prodejním týmem, pak dokud nedojde k zaslání informací o objednávce oddělení materiálu, nemůže toto oddělení na novou zakázku reagovat. Tím dochází k časovým prodlevám. Dalším příkladem je i vytváření nové zakázky, kdy prodejní oddělení musí získávat aktuální data o dostupných množstvích materiálů od oddělení materiálu, případně od více oddělení materiálů. (Bandari, 2016)

Výhodou ERP systémů v této oblasti je jednotná databáze, která je zobrazena na obrázku 2.2. Jednotlivá oddělení tedy nemusí komunikovat mezi sebou, ale mají k dispozici veškerá aktuální data všech oddělení. (Bandari, 2016)

Obrázek 2.2 Databáze v ERP systému



Zdroj: (Bandari, 2016) – vlastní zpracování

2.2 Historický vývoj ERP systémů

Pojem ERP byl zaveden v devadesátých letech dvacátého století, ale historie těchto systémů začala již před více než sto lety.

Roku 1913 vytvořil Ford Whitman Harris papírově založený výrobní systém, který se stal pod názvem Economic Order Quantity standardem pro řízení výroby a po desítky let jím také zůstal. (www.oracle.com)

Významná změna nastala roku 1964, kdy společnost zabývající se výrobou nářadí Black and Decker jako první na světě zkombinovala Economic Order Quantity model a sálový počítač. Nejednalo se však o ERP systém, jak je znám nyní, ale o Materials Requirements Planning (MRP) systém, v překladu systém pro plánování zásob. Prostřednictvím tohoto typu systému bylo možno řídit množství zásob, a tím šetřit náklady vycházející z nadbytečného množství zásob na skladě. (www.oracle.com)

V sedmdesátých letech dvacátého století využívaly MRP systémy již stovky společností, a to zejména největší výrobní společnosti této doby, jelikož správa takového softwaru byla v té době velice nákladná. (www.versaccounts.com)

Během osmdesátých let minulého století došlo k pokroku ze systémů MRP na systémy MRP II. Rozdíl mezi těmito dvěma typy systémů představoval zejména přidání více funkcionalit v oblasti výrobních procesů. Namísto pouhého plánování zásob mají MRP II systémy druhé generace funkcionality, jako jsou například plánování výroby, nákup součástek, pokročilé řízení zásob apod. (https://www.erp-information.com)

V devadesátých letech minulého století se poprvé začíná užívat termín ERP a z řízení materiálových zásob se postupně stalo řízení celopodnikových zdrojů. ERP systémy začaly svými funkcionalitami pokrývat kromě materiálů veškeré aktivity

společností, ať už se jedná o oblast strojírenství, lidských zdrojů, účetnictví, projektového řízení a dalších oblastí. (www.versaccounts.com)

Roku 1996 vytvořila společnost Netsuite ERP systém, který jako první nemusel být provozován na vlastní infrastruktuře, ale bylo možné se k němu připojit prostřednictvím Internetu. Tento krok znamenal vznik nové generace tzv. ERP II systémů, které poskytovaly přístup k ERP funkcionalitám v reálném čase. Také došlo k přidání nových funkcionalit těchto systémů, kterými byly řízení vztahu s dodavatelem, řízení vztahu se zákazníkem a podpora analytických a reportingových nástrojů. (www.versaccounts.com)

Po roce 2005 byl zaznamenán nárůst popularity cloudových řešení. V této době došlo k velkému posunu od tradičního přístupu ke cloudovému, díky kterému mohly společnosti využívat výhody ERP systémů bez vysokých pořizovacích a provozních nákladů. Tato možnost je ideální pro malé a střední firmy, jelikož poskytuje veškeré funkcionality ERP systémů za mnohem nižší cenu. (www.versaccounts.com)

V dnešní době se ERP systémy snaží podpořit získání co největší konkurenční výhody pro danou společnost. K tomu dochází využíváním nejmodernějších technologií a přístupů, ať už využitím analýzy big data, strojového učení, umělé inteligence nebo virtuální reality. Veškeré kroky vedou k podpoře automatizace výroby, snížení nákladů a zvýšení zisku. (www.erp-information.com)

2.3 Platforma SAP ERP

Společnost SAP má největší podíl na trhu ERP softwaru a její softwarové systémy jsou v současné době používány tisíci zákazníky. ERP systém pak poskytuje aplikace napříč všemi oblastmi společností. (www.businessnsoftware.com)

2.3.1 Historický vývoj systémů SAP

Systém SAP R/1 je první systém společnosti SAP, který byl uveden na trh v roce 1972 a byl určen k vedení účetnictví. V názvu systému pak písmeno R znamená zpracování dat v reálném čase a jednička značí jednovrstvou architekturu. (Bandari, 2016)

Systém SAP R/2 byl uveden na trh koncem sedmdesátých let minulého století. Z jeho názvu je patrné, že architektonicky se jednalo o systém dvouvrstvý, přičemž jedna

vrstva byla prezentační a druhá vrstva se skládala z aplikační a databázové části. (Bandari, 2016)

Systém SAP R/3 představoval posun k třívrstvé architektuře. Byl představen v devadesátých letech minulého století a skládal se z prezentační, aplikační a databázové vrstvy, kde všechny vrstvy byly od sebe vzájemně odděleny. (Bandari, 2016)

Systém SAP R/3 4.70 byl první systémem této řady, který obsahoval platformu SAP webového aplikačního serveru umožňujícího vytvářet Business Server Pages aplikace a integrujícího J2EE engine, pomocí kterého je možné na aplikačním serveru provádět programy psané nejen v programovacím jazyce ABAP, ale i Java. Zkratka J2EE znamená Java 2 Enterprise Edition a představuje rozšíření standardní edice Java např. pro webové služby. (www.theserverside.com)

SAP webový aplikační server je nutností k podpoře webové infrastruktury. Technologie Business Server Pages je přitom obdobou technologií JavaServer Pages společnosti Sun Microsystems a Active Server Pages společnosti Microsoft. (Bandari, 2016)

Systém SAP R/3 Enterprise 4.70 se zaměřoval zejména na SAP webový aplikační server, v jehož rámci došlo k přidání nových funkcionalit a rozdělení komponent systému dle funkčních oblastí s možností implementace vlastního rozšíření pro tyto funkcionality. (Bandari, 2016)

Systém mySAP ERP 2004 ECC 6.0 ERP 6.0 skýtal mnohé další architektonické změny. Bylo v něm realizováno oddělení technologické a aplikační části systému, kde aplikační část představoval SAP Enterprise Central Component a technologickou základnu pak SAP Netweaver. (Bandari, 2016)

Verze systémů SAP jsou spolu s rokem jejich vydání vypsány v tabulce 2.1.

Tabulka 2.1 Verze systémů SAP

Rok vydání	**Verze**
1972	SAP R/1
Konec sedmdesátých let 20. století	SAP R/2
1992	SAP R/3
2001	Představen SAP Webový Aplikační server
2002	SAP R/3 Enterprise 4.70
2004	mySAP ERP 2004 ECC 5.0 ERP 5.0 (SAP Webový Aplikační Server 6.40, SAP NetWeaver 2004)
2005	mySAP ERP 2005 ECC 6.0 ERP 6.0 (SAP Webový Aplikační Server 7.00; NetWeaver 2004 s/7)
2006	SAP NetWeaver 7.1
2011	SAP NetWeaver 7.3
2013	SAP NetWeaver 7.4
2015	SAP NetWeaver 7.5

Zdroj: (Bandari, 2016) – vlastní zpracování

Roku 2010 došlo k uvedení na trh databázového systému SAP HANA, která je součástí platformy SAP HANA. (www.news.sap.com)

Systémy využívající platformu SAP HANA, tedy zejména systémy SAP, jsou v aktuální době racionálním krokem před přechodem na systém SAP S/4 HANA, který byl uveden roku 2015 a pro jeho používání není možné využívat jinou databázi než SAP HANA. (www.mulesoft.com)

Rozšiřující balíčky systému SAP slouží k přidávání nových funkcionalit bez nutnosti zásahů do celého systému SAP. Balíčky obsahují jak nové funkcionality, tak optimalizaci stávajících programů pro platformu SAP HANA. Prvním balíčkem obsahujícím funkcionality optimalizované pro platformu SAP HANA pro ERP 6.0 byl

EHP 6. Rozšiřující balíčky jsou spolu s rokem jejich vydání vypsány v tabulce 2.2. (Bandari, 2016)

Tabulka 2.2 Rozšiřující balíčky systému SAP

Rok vydání	**Název rozšiřujícího balíčku**
2006	EHP 1
2007	EHP 2
2007	EHP 3
2008	EHP 4
2010	EHP 5
2011	EHP 6
2014	EHP 7
2016	EHP 8

Zdroj: (Bandari, 2016) – vlastní zpracování

2.3.2 Architektura systému SAP

Jak již bylo zmíněno, architektura systému SAP se postupně z jednovrstvého systému vyvinula v třívrstvou. Třívrstvá architektura se skládá z prezentační, aplikační a databázové vrstvy. (Kühnhauser, 2009)

Prezentační vrstva

Prezentační vrstva se skládá z uživatelského rozhraní, které běží na zařízení uživatele a využívá jeho grafické uživatelské rozhraní. Jelikož může být prezentační vrstva vyvinuta bez závislosti na operačním systému koncového zařízení uživatele, je zde podporováno velké množství koncových uživatelských zařízení, ať už se jedná o počítače nebo chytré telefony. (Kühnhauser, 2009)

Aplikační vrstva

Aplikační vrstva je určena pro běh programů. Tato vrstva se může skládat z více aplikačních serverů, které jsou koordinovány prostřednictvím tzv. messaging serveru. (Kühnhauser, 2009)

Databázová vrstva

Databázová vrstva slouží zejména k uložení veškerých persistentních dat. Databáze je obvykle řízena pomocí systému řízení báze dat. (Kühnhauser, 2009)

2.3.3 SAP Moduly

Z důvodu robustnosti a komplexity celkového systému je pro získání vyšší flexibility a ulehčení customizace celý systém rozdělen do jednotlivých programových modulů. (Bandari, 2016)

Každý programový modul pak pokrývá určité oblasti společností (např. SAP ERP MM module pokrývá oddělení materiálu). Navíc je možné všechny SAP moduly integrovat s jinými programovými systémy, případně i programovými systémy třetích stran. (Bandari, 2016)

SAP moduly se dělí do dvou oblastí, a to funkcionální a technické. Toto rozdělení je možno uplatnit i na SAP konzultanty, kdy funkcionální konzultanti jsou zaměřeni zejména na procesní a odbornou oblast daného modulu a techničtí konzultanti jsou pak zaměřeni obecně na oblasti programování, práci s databázovými systémy a operační systémy. (www.solutiondots.com)

Funkcionální moduly pokrývají procesy a funkční oblasti obchodních aktivit, přičemž každý z modulů má i svou dvoupísmenkovou zkratku. Funkcionální moduly a jejich zkratky jsou zobrazeny v tabulce 2.3.

Tabulka 2.3 Funkcionální moduly a jejich zkratky

Název funkcionálního modulu	**Zkratka funkcionálního modulu**
Financial Accounting	FI
Controlling	CO
Human Resource	HR
Sales and Distribution	SD
Materials Management	MM
Plant Maintenance	PM
Production Planning	PP

Zdroj: (Bandari, 2016) – vlastní zpracování

Technické moduly pokrývají technickou stránku SAP systémů a jsou potřebné pro úpravy systému na základě požadavků společnosti. Technické moduly jsou vypsány v tabulce 2.4.

Tabulka 2.4 Technické moduly

Název technického modulu
SAP Basis
ABAP
SAP BusinessObjects Business Intelligence (BI)/SAP Business Warehouse (BW)
SAP Exchange Infrastructure (XI)/SAP Process Integration (PI)
SAP Enterprise Portal
SAP HANA

Zdroj: (Bandari, 2016) – vlastní zpracování

2.3.4 Změny v systému SAP

Systém SAP obsahuje velké množství předem vytvořených programů, které pokrývají obecné procesy řešené jednotlivými odděleními společností. Avšak každá společnost může mít specifické požadavky. Tyto požadavky jsou pak řešeny prostřednictvím tzv. customizace systému. Customizace je možné dosahovat třemi způsoby. (Bandari, 2016)

Modifikace

Při modifikaci se jedná o změnu zdrojového kódu SAP programu.

Tento způsob je nevhodný zejména kvůli tomu, že při upgradu systému SAP na jeho vyšší verzi dojde vždy ke zrušení této změny. V případě velkého množství modifikací je tedy velice náročné zajistit u systému s provedeným upgradem, aby všechny modifikace byly opětovně implementovány. Navíc v případě, že systém po modifikaci nebude fungovat očekávaným způsobem, není možné očekávat podporu ze strany společnosti SAP.

Customizace prostřednictvím modifikací je tedy nedoporučované řešení, které může způsobovat velké potíže. (Bandari, 2016)

Zákaznické vylepšení (Customer Enhacement)

Při zákaznickém vylepšení nedochází ke změně zdrojového kódu SAP programu, ale k napojení SAP programu a programu získaného vlastním vývojem.

Jelikož je očekáváno, že jednotlivé společnosti budou chtít systémy SAP upravit dle vlastních specifikací, poskytují programové systémy společnosti SAP možnosti zákaznického vylepšení. Jedná se například o situaci, kdy při vytvoření objednávky může každá společnost jiným způsobem kontrolovat zadané informace o objednávce. SAP programy proto například poskytují možnost implementovat metodu programu, která specifickým způsobem zadané hodnoty kontroluje. (Bandari, 2016)

Vlastní vývoj

Pokud neexistuje SAP program, který danou problematiku řeší, případně jej není možné vhodně modifikovat využitím doporučených postupů společností SAP, je pak nutné vytvořit vlastní programové řešení.

Při jeho implementaci je pak zejména důležité dodržování jmenných konvencí. Názvy programů vytvořených vlastním vývojem je nutné začínat písmeny Z nebo Y. Případně je nutné rezervovat jmenný prostor u společnosti SAP. Pokud by tato konvence nebyla dodržena, mohlo by v případě upgradu systému dojít k tomu, že by byl implementován program společností SAP, který by přepsal program vytvořený vlastním vývojem. (Bandari, 2016)

2.3.5 Typy aplikací v systému SAP

Mezi základní typy aplikací, které lze vyvíjet, patří reporty, rozhraní, konverze, rozšíření a formuláře. (Bandari, 2016)

Reporty

Reporty poskytují přístup k databázi pouze pro čtení. Jazyk ABAP byl dříve nejvíce využívám zejména pro psaní reportů. Jedná se v principu o jednoduché získání dat z databáze a zobrazení výstupu na obrazovce. (Bandari, 2016)

Rozhraní

Tyto programy slouží k výměně dat s jinými programovými systémy. Obecně existují dva typy těchto rozhraní, a to vstupní a výstupní. Vstupní rozhraní slouží pro získávání dat z jiných systémů, výstupní rozhraní slouží pro zasílání dat ze systému do jiných programových systémů. (Bandari, 2016)

Konverze

Konverze jsou programy, které slouží k migraci dat z jednoho systému do jiného. Jsou využity jednorázově při přechodu na novou verzi systému. (Bandari, 2016)

Rozšíření

Mezi rozšíření patří prováděné změny v programech SAP, zákaznická vylepšení, vlastní vývoj a připojené programy třetích stran. (Bandari, 2016)

Formuláře

Obdobně jako reporty fungují formuláře ve smyslu poskytování přístupu k databázi pouze pro čtení. Avšak formuláře jsou využívány v případech, kdy není cílem zobrazit data na výstupní obrazovce, ale vytvořit jeho fyzickou verzi. Jsou tedy využity například tehdy, kdy je potřeba vytisknout fakturu. (Bandari, 2016)

2.4 Programovací jazyk ABAP

Advanced Business Application Programming (ABAP – v překladu programování pokročilých firemních aplikací) je programovací jazyk využívaný pro vlastní implementaci aplikací v systému SAP. (Kühnhauser, 2009)

Jedná se o hybridní programovací jazyk, což znamená, že umožňuje vytvářet programy jak s využitím procedurálního, tak i objektově orientovaného paradigmatu programování. (Kühnhauser, 2009)

2.4.1 Historie programovacího jazyka ABAP

Programovací jazyk ABAP byl uveden na trh v osmdesátých letech dvacátého století. Zpočátku byl určen jako programovací jazyk k tvorbě reportů pro dvouvrstvý systém SAP R/2 a využíván v oblastech finančního řízení, správy materiálu a účetnictví. (www.cleverism.com)

S příchodem třívrstvého systému SAP R/3 a rozšíření oblastí využití systému SAP se stal programovací jazyk ABAP nejpoužívanějším jazykem pro tvorbu programů pro systémy SAP. (www.cleverism.com)

Za dobu své existence prošel tento programovací jazyk mnoha změnami a vylepšeními, aby se udržel v konkurenci moderních trendů v oblasti informačních technologií. Za jednu z největších změn lze považovat uvedení rozšíření ABAP Object,

které roku 1999 umožnilo psát programy v jazyce ABAP jiným než typicky procedurálním způsobem. (www.cleverism.com)

2.4.2 Analýza běhu programu

Programy implementované v programovacím jazyce ABAP je možné analyzovat podle řady kritérií, ať už se jedná o jejich časovou náročnost při získání dat z databází, nebo rychlost běhu programu. Základní nástroje pro analýzu běhu programu jsou vypsány v tabulce 2.5. (Bandari, 2016)

Tabulka 2.5 Základní nástroje pro analýzu běhu programu

Nástroj	**Popis**
ABAP Unit	ABAP Unit je nástroj pro vývoj testů pro programy v jazyce ABAP.
Code Inspector	Slouží ke kontrole zdrojového kódu při kompilaci.
Memory Inspector	Je součástí ABAP Debuggeru a slouží k analýze využití paměti při běhu programu.
Performance Trace/ Runtime analysis	Pomocí tohoto nástroje je možné analyzovat, jak dlouho trvá provedení vzdáleného volání funkce, případně jak dlouho trvá provedení dotazu na databázi.
Single-transaction analysis	Nástroj slouží k analýze běhu programu, oproti nástroji Performance Trace/ Runtime Analysis poskytuje rozsáhlejší možnosti analýzy.
Dump analysis	Pokud dojde k ukončení programu v důsledky vzniku programové výjimky, dojde k vygenerování zprávy o selhání, kterou je poté možno analyzovat a zjistit příčinu ukončení programu.

Zdroj: (Bandari, 2016) – vlastní zpracování

2.5 Platforma SAP HANA

V jádru je SAP HANA moderní relační databáze, která je optimalizovaná jak pro analytické, tak i transakční případy užití. Název HANA představuje zkratku pro High Performance Analytical Appliance, neboli vysoce výkonné analytické zařízení. Je důležité zmínit, že SAP HANA není pouze relační databáze, ale že tento název zaštiťuje skupinu produktů, které jsou uvedeny níže. Existuje také velké množství dalších jejich nástrojů a rozšíření. (Gahm, 2016)

Databázový systém SAP HANA

Relační databáze SAP HANA poskytuje funkcionality obdobné tradičním relačním databázím, které jsou podporovány systémem SAP. Stejně jako tyto tradiční relační databáze, poskytuje SAP HANA funkcionality pro zálohování a obnovu, podporuje SQL standard a zajišťuje konzistenci dat při transakcích dodržováním principů ACID **(**www.guru99.com).

Na rozdíl od ostatních relačních databází dokáže SAP HANA umístit všechna důležitá data do hlavní paměti počítače. Kombinací řádkových, sloupcových a na objektech založených databázových technologií a spolu s optimalizací na paralelní zpracování dat je možno plně využít potenciál vícejádrových a víceprocesorových architektur. (Gahm, 2016)

SAP HANA Studio

SAP HANA Studio představuje zejména administrativní a vývojové prostředí. Pomocí tohoto nástroje je možné například spouštět nebo zastavovat databázové služby, monitorovat systém, měnit nastavení systému, spravovat uživatele a jejich oprávnění. (Baumgartl, Bardhan, 2018)

SAP HANA Klient

Pomocí SAP HANA Klientu je možné navázat spojení s databází SAP HANA prostřednictvím vybraného síťového protokolu. (Gahm, 2016)

SAP HANA XS

SAP HANA XS, kde XS znamená Extended Application Services, poskytuje možnost vyvíjet aplikace přímo v prostředí SAP HANA bez nutnosti zapojení dalšího aplikačního serveru. SAP HANA XS poskytuje kompletní vývojové prostředí sestávající z webového serveru, vývojového modelu a podpůrných nástrojů. Aplikace jsou vyvíjeny

pomocí několika vývojových objektů, které pokrývají vše od definice datového modelu až po vytváření uživatelských rozhraní. (Gahm, 2016)

2.6 Paralelní zpracování dat

Programy, které zpracovávají velké množství dat, by měly být schopny zpracovat data paralelně tak, aby se flexibilním způsobem adaptovaly na dané hardwarové vybavení a vhodně využily jeho zdroje. (Gahm, 2009)

Před paralelizací programu by mělo být ověřeno, že běh programu nelze optimalizovat jinými způsoby. Nejprve je tedy nutné analyzovat běh stávajícího programu a optimalizovat jej, pokud i poté je jeho výkon nedostatečný, je vhodné využít možnosti paralelizace programu.

Při paralelizaci je nutno brát v potaz určité problematické oblasti, které mohou nastat. Jednotlivé oblasti jsou vypsány níže. (Gahm, 2009)

2.6.1 Paralelizovatelné oblasti běhu programu

Při návrhu paralelizace programu je nutno si uvědomit, které části programu mohou běžet paralelně a které nemohou. Je nutné zvážit, jestli jsou jednotlivě prováděné kroky programu závislé na svém pořadí a zdali by změna tohoto pořadí neovlivnila negativně běh programu. Vhodné je, aby paralelizované části programů mohly běžet nezávisle na sobě, případně je nutno zabezpečit, že provedenou paralelizací nedojde k narušení správného běhu programu. (Gahm, 2009)

2.6.2 Zámky

Paralelní procesy mohou nastavovat zámky na objekty, se kterými potřebují komunikovat. Čekání jiných procesů na tyto zámky může výrazně zpomalit běh programu. Jedná se zejména o zámky na SAP úrovni a na databázové úrovní. (Gahm, 2009)

Zámky na SAP úrovni jsou explicitně nastaveny vývojářem a v jejich případě je také možnost nastavit, co by se mělo stát v případě, kdy je za běhu programu potřeba komunikovat s uzamčeným objektem. Zejména je pak nutno nastavit časový interval, v jakém by se měl daný proces znovu pokusit komunikovat s dříve uzamčeným objektem. (Gahm, 2009)

Zámky na databázové úrovni jsou implicitně nastaveny při provádění změn záznamů v databázi. V případě, že daný proces potřebuje nastavit takový zámek, ale tento již je nastaven, pak daný proces čeká na uvolnění zámku. (Gahm, 2009)

2.6.3 Deadlock

Deadlock nastává, pokud se procesy uzamknou navzájem při svém běhu, tj. např. pokud jeden proces čeká na objekt uzamknutý druhým procesem a současně druhý proces čeká na objekt uzamknutý prvním procesem a žádný z těchto procesů přitom neuvolní jím držený zámek objektu. (Gahm, 2009)

2.6.4 Velikosti pracovních balíčků

Při využití paralelního zpracování je snaha o rozložení velkého množství dat do pracovních balíčků, které jsou pak zpracovány paralelně. Je nutné dbát na to, aby jejich velikosti byly úměrné. Pokud by jeden z pracovních balíčků obsahoval znatelně více záznamů než ostatní balíčky, pak by celková doba běhu programu byla zdržena tímto balíčkem, zatímco ostatní procesy s méně záznamy by na tento proces musely čekat. (Gahm, 2009)

2.6.5 Přerušené zpracování pracovních balíčků

Pokud by systém během zpracovávání pracovního balíčku rozhodl o zrušení zpracovávání, pak je nutné definovat, zdali je možno se opětovně pokusit toto zrušené zpracování opakovat, nebo jestli je nutné provést nějaké opravné kroky. (Gahm, 2009)

2.6.6 Rovnoměrné využití hardwaru

Při rozložení balíčků na hardware je nutné dbát na rovnoměrné využití více aplikačních serverů, jinak by mohlo dojít k přehlcení jednoho serveru a nevyužití ostatních. (Gahm, 2009)

2.7 Možnosti paralelního zpracování v jazyce ABAP

V současné době jsou k dispozici dvě základní metodiky, jejichž aplikací lze docílit paralelního běhu programu. Jedná se o metodiku statického využití procesů na pozadí a metodiku dynamického využití asynchronních RFC.

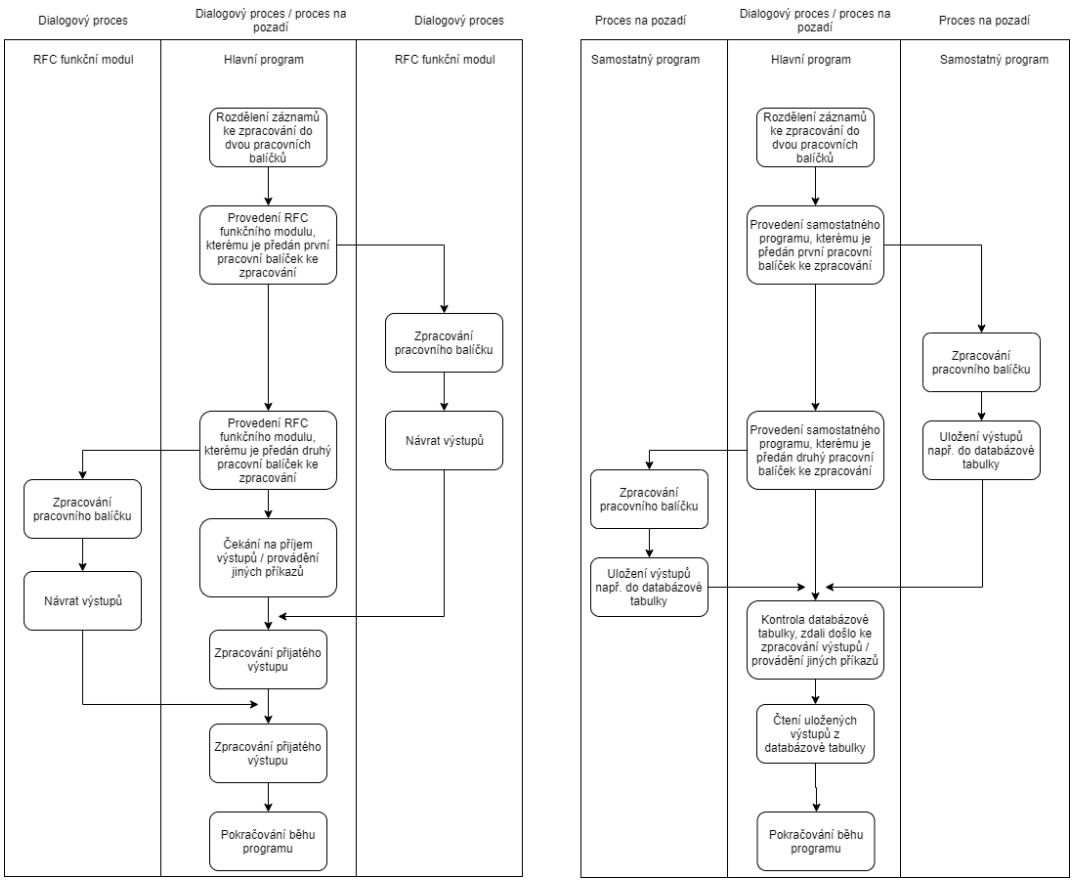
Zásadní rozdíl mezi těmito metodikami je ten, že při využití metodiky dynamického využití asynchronních RFC jsou při paralelním zpracování dat využívány dialogové pracovní procesy, tedy pracovní procesy, které se využívají např. ke zpracování požadavků uživatele. Při využití metodiky statického využití procesů na pozadí jsou

využívány procesy na pozadí, které se využívají zejména pro běh dlouhotrvajících, periodicky se opakujících programů.

Dalším zásadním rozdílem obou metodik je možnost získání výstupů. Při použití metodiky dynamického využití asynchronních RFC jsou prováděny funkční moduly a předání výsledků k dalšímu zpracování je umožněno nastavit před provedením funkčního modulu. Avšak při zvolení metodiky statického využití procesů na pozadí jsou prováděny samostatné programy a k předání výsledku k dalšímu zpracování je nutno využít např. databázové tabulky.

Zjednodušené znázornění obou metodik je zobrazeno na obrázku 2.3. Na levé straně je zobrazena metodika dynamického využití asynchronních RFC a na pravé straně je zobrazena metodika statického využití procesů na pozadí.

Obrázek 2.3 Znázornění metodik paralelního zpracování dat



Zdroj: vlastní zpracování

2.7.1 Metodika statického využití procesů na pozadí

Pokud by bylo cílem paralelizovat běh programu, který například zpracovává údaje o všech uživatelích systému identifikovaných prostřednictvím unikátního

identifikačního klíče, bylo by možné rozdělit záznamy podle klíče do několika skupin a následně načasovat a spustit program na pozadí s danou skupinou záznamů. Poté by se program musel dotazovat tabulky **`TBTCO`** (tabulka pro přehled stavů procesů), zdali veškeré programy spuštěné na pozadí ukončily svůj běh, následně pak získat zpracovaná data a pokračovat v synchronním zpracování. (Gahm, 2009)

Tato metodika je vhodná zejména pro dlouhotrvající procesy, které mohou být rozděleny do statických pracovních balíčků.

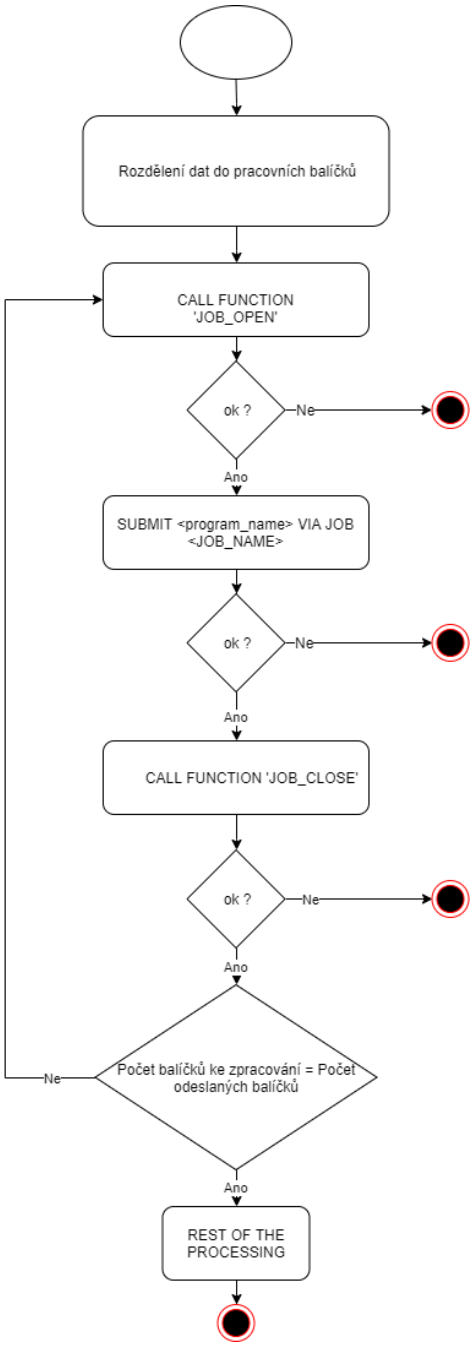
Postup implementace paralelního zpracování dat využitím procesů na pozadí

Prvním krokem je identifikovat část programu, která by měla být zpracována paralelně. Je nutné dbát na to, aby při využití procesů na pozadí nedocházelo po ukončení běhu procesu na pozadí k navrácení se k původnímu programu, který proces na pozadí vytvořil. Výhodou je možnost nastavení plánovaného začátku spuštění procesu na pozadí, který nemusí být zahájen okamžitě.

K implementaci této strategie jsou využity standardní funkční moduly obsahující programové procedury vytvořené společností SAP k zabezpečení určitých funkcionalit, které je možné využívat z různých programů a dle oblasti využití jsou seskupovány do funkčních skupin. Tyto funkční moduly poslouží k vytvoření a naplánování běhu programu v procesu na pozadí.

Postup implementace je zobrazen na obrázku 2.4 a následně popsán níže.

Obrázek 2.4 Postup implementace – Statická metodika



Zdroj: vlastní zpracování

Při paralelním zpracováním dat prostřednictvím procesů na pozadí je vhodné využít standardní funkční moduly **`JOB_OPEN`** a **`JOB_CLOSE`**.

Prvním krokem implementace je rozdělení dat, která by měla být zpracována paralelně do pracovních balíčků, které budou jednotlivé procesy na pozadí zpracovávat. Vhodný počet pracovních balíčků může být obtížné určit, jelikož při nízkém počtu bude paralelní běh omezen na malý počet paralelních běhů, avšak při rozdělení do mnoha

pracovních balíčků může dojít k omezení jiných programů, které běží prostřednictvím procesů na pozadí ve stejný čas, jelikož veškeré zdroje budou alokovány danému programu.

Druhý krokem je pak provedení funkčního modulu **JOB_OPEN**, kterému je jako parametry možno předat název procesu na pozadí, kterým bude běh identifikován.

Třetím krokem je nastavení programu, který má být v rámci procesu na pozadí spuštěn, a předání argumentů, se kterými má být tento program proveden.

Čtvrtým krokem je naplánování času, kdy má být daný program spuštěn, např. může být naplánováno okamžité spuštění, případně opožděné nebo periodicky se opakující spouštění ve zvoleném časovém intervalu.

2.7.2 Metodika dynamického využití asynchronních RFC

Asynchronní RFC (zkratka RFC znamená Remote Function Call, neboli asynchronní vzdálené volaní funkce) se v jazyce ABAP využívá pro asynchronní vzdálené volaní funkčního modulu. (www.docs.oracle.com)

Asynchronně vzdáleně volané funkční moduly jsou funkční moduly, které je možno provádět vzdáleně, ať už interně v rámci daného systému SAP, nebo externě mimo systém SAP. Při provádění asynchronně vzdáleně volaného funkčního modulu dochází k využití nového dialogového procesu, pomocí kterého je možno zpracovávat data paralelně a volající program nečeká na dokončení asynchronně vzdáleně volaného funkčního modulu. (www.docs.oracle.com)

Využití interních asynchronně vzdáleně volaných funkčních modulů je možno ve zdrojovém kódu programu realizovat prostřednictvím příkazu **STARTING NEW TASK**, po jehož provedení může běh programu pokračovat a nemusí čekat na dokončení asynchronně vzdáleně volaného funkčního modulu. (Wood, 2010)

Při volání tohoto asynchronně vzdáleně volaného funkčního modulu může být definována procedura nebo metoda ve volajícím programu, která bude po dokončení asynchronně vzdáleně volaného funkčního modulu provedena pomocí příkazu **ON END OF TASK.** (Wood, 2010)

Běh programu, který volá asynchronně vzdáleně volaný funkční modul, může být po dobu běhu asynchronně vzdálených funkčních modulů pozastaven pomocí příkazů **WAIT FOR ASYNCHRONOUS TASKS**. (Wood, 2010)

Pokud je voláno více asynchronně vzdáleně funkčních modulů, běží tyto funkční moduly obecně paralelně a vždy v dialogovém režimu, tedy na popředí (oproti zmíněné metodice paralelizace využitím procesů na pozadí). (Gahm, 2009)

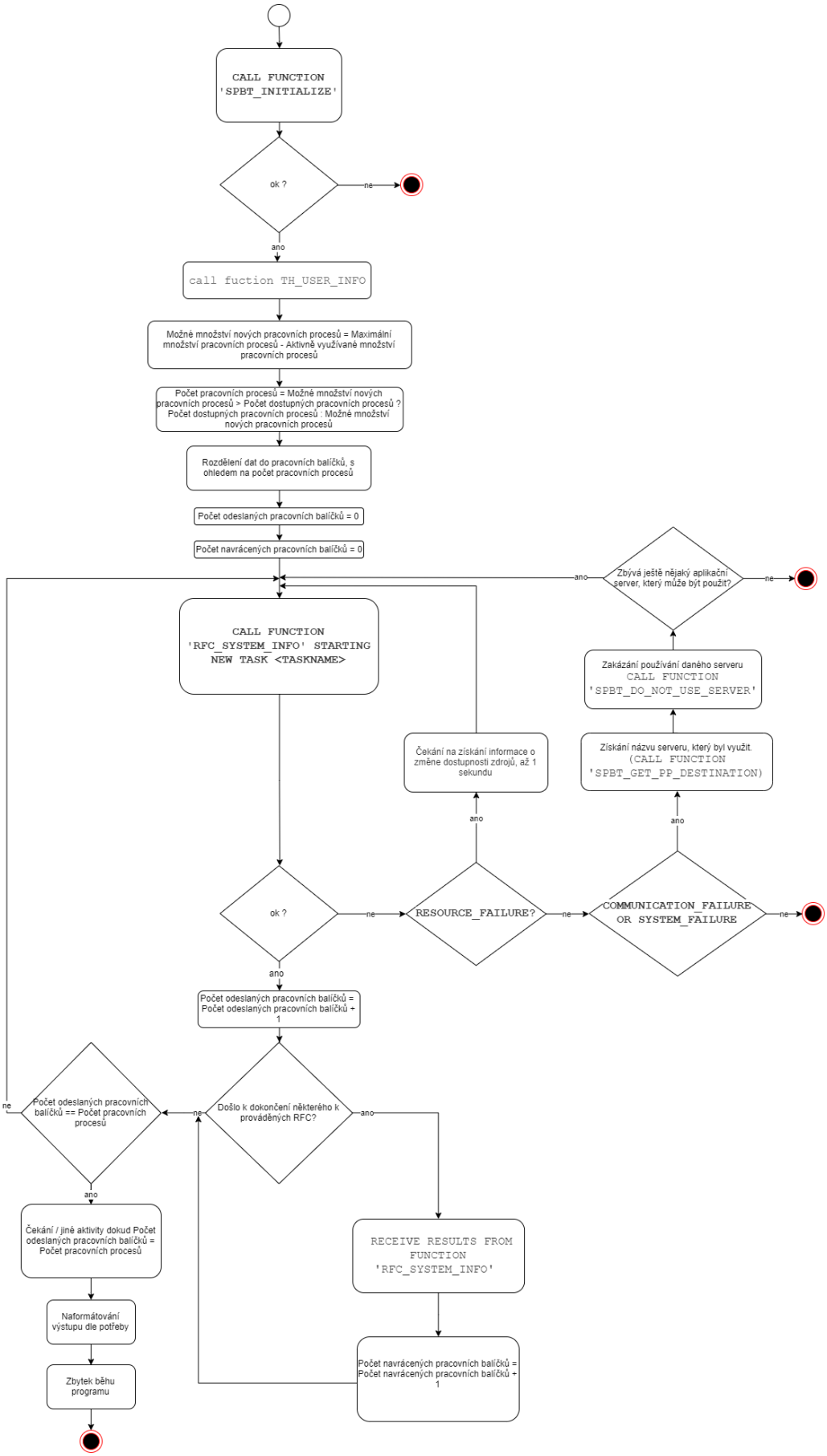
Postup implementace paralelního zpracování dat využitím asynchronních RFC

Nejprve je nutno identifikovat část programu, která má být zpracována paralelně. Tato část musí být prováděna v asynchronně vzdáleně volaném funkčním modulu. Pro ten je specifické zejména to, že asynchronně vzdáleně volané funkční moduly mohou být volány nejen prostřednictvím jazyka ABAP, a proto není možné předávat tomuto funkčnímu modulu parametry jako referenční odkazy, ale pouze jako hodnoty.

Při volání asynchronně vzdáleně volaných funkčních modulů je rovněž možno specifikovat skupinu aplikačních serverů, které mají daný požadavek zpracovat. Výstupy asynchronně vzdáleně volaného funkčního modulu je možné získat pomocí definované procedury nebo metody, která je zavolána po dokončení asynchronně vzdáleně volaného funkčního modulu. (Wood, 2010)

Postup implementace je zobrazen na obrázku 2.5 a následně popsán níže.

Obrázek 2.5 Postup implementace – Dynamická metodika



Zdroj: vlastní zpracování

Prvním krokem v rámci implementace je provedení funkčního modulu **SPBT_INITIALIZE.** Tento funkční modul slouží ke kontrole, že zvolená skupina aplikačních serverů je validní, a rovněž ke zjištění počtu pracovních procesů, které jsou k dispozici. Uvedený funkční modul sice není nutné explicitně provádět, neboť pokud tento funkční modul není explicitně proveden, provede jej systém SAP implicitně, ale v rámci programu již pak není možno pracovat s množstvím volných pracovních procesů. Pokud provedení tohoto funkčního modulu selže, není možno pokračovat v dalším provádění daného programu. (Wood, 2010)

Druhým krokem postupu je rozdělení dat do pracovních balíčků, které budou paralelně zpracovávány. Toto rozdělení je možno přizpůsobit počtu volných procesů. Důležité je zkontrolovat, kolik má uživatel již spuštěných dialogových pracovních procesů a jaký je maximální povolený počet procesů na jednoho uživatele. Přitom je potřeba pracovat s nižším počtem z celkového počtu dostupných dialogových pracovních procesů pro daného uživatele a počtu dostupných dialogových pracovních procesů ve skupině serverů. Tento počet představuje vhodný počet pracovních balíčků, které budou zpracovávány. (Gahm, 2009)

Třetím krokem postupu je samotné volání asynchronně vzdáleně volaného funkčního modulu. Každé volání asynchronně vzdáleně volaného funkčního modulu má specifikovaný název úlohy, který slouží k identifikaci daného volání. (Wood, 2010)

Pokud při volání asynchronně vzdáleně volaného funkčního modulu nastane výjimka typu **COMMUNICATION_FAILURE** při komunikaci s aplikačním serverem, nebo systémová výjimka typu **SYSTEM_FAILURE**, je pak nutné zjistit, jaký aplikační server ze skupiny aplikačních serverů byl použit, a následně jej vymazat ze seznamu používaných aplikačních serverů, aby na něj nebyly zasílány další požadavky. Pokud by výmazem aplikačního serveru ze skupiny aplikačních serverů došlo k tomu, že by nebyl k dispozici již žádný aplikační server, není možno dále pokračovat v provádění programu. Původní požadavek musí být tedy opakovaně realizován voláním asynchronně vzdáleně volaného funkčního modulu. (www.help.sap.com)

Pokud při volání asynchronně vzdáleně volaného funkčního modulu nastane výjimka typu **RESOURCE_FAILURE,** znamená to, že již není dostupný žádný pracovní proces, který by mohl zpracovat požadavek. V této situaci je pak nutno počkat, až některý

z již spuštěných asynchronně vzdáleně volaných funkčních modulů dokončí svůj běh a v důsledku této skutečnosti se stane pracovní proces, který jej zpracovával, znovu dostupným, nebo až uběhne určitý časový interval, po jehož ukončení by se mohl ve skupině serverů uvolnit pracovní proces využívaný jiným než daným programem. Poté je nutné opět opakovat původní požadavek.

Po dokončení běhu asynchronně vzdáleně volaného funkčního modulu může dojít k získání výsledků prostřednictvím zvolené procedury nebo metody provedením příkazu **`RECEIVE RESULTS`**. (www.help.sap.com)

Třetí krok postupu se pak znovu opakuje, dokud nejsou všechny pracovní balíčky odeslány.

Čtvrtým krokem postupu je čekání na dokončení všech asynchronně vzdáleně volaných funkčních modulů. Je přitom důležité brát v úvahu, že asynchronně vzdáleně volané funkční moduly jsou zpracovávány jako dialogové procesy, a nikoliv jako procesy na pozadí. Proto je nutné brát v potaz systémovou proměnnou, která limituje dobu, po kterou může být dialogový pracovní proces souvisle využíván. (www.help.sap.com)

Pátým krokem postupu je zpracování výstupů. Jedná se pak zejména o jejich případně seřazení, jelikož při využití asynchronně vzdáleně volaných funkčních modulů není garantováno pořadí, v jakém dojde k návratu a zpracování daného provedení modulu. To je dáno zejména tím, že je využíváno více aplikačních serverů, které jsou instalovány v rámci jedné skupiny serverů, a tyto aplikační servery mohou být přitom rozdílné.

2.8 Návrhové vzory a jejich využití v jazyce ABAP

Když začne architekt plánovat stavbu nové budovy, tak nevynalézá znovu kolo, ale použije časem ověřené a otestované postupy a návrhy. (Koseoglu, 2016)

To stejné platí pro softwarové architekty. Objektově orientované paradigma programování poskytuje mnoho konceptů, které je možno znovu využít. Návrhové vzory poskytují jednoduché, flexibilní a rozšiřitelné řešení na časté požadavky a problémy v této oblasti. (Koseoglu, 2016)

Prvním komplexně zdokumentovaným zdrojem v této oblasti je monografie Design Patterns Elements of Reusable Object-Oriented Software, kterou napsal “Gang čtyř” (Erich Gamma, Richard Helm, Ralph Johnson a Johna Vlissidese). I přes to, že byla

tato kniha vydána v roce 1994, je mnoho dnešních aplikací založeno na konceptech definovaných v této monografii. (Koseoglu, 2016)

Známých návrhových vzorů je v současné době k dispozici celá řada a mohou být členěny do různých skupin. Obvykle jsou rozdělovány na architectural, creational, structural a behavioral. V překladu architektonické, tvořící, strukturální a behaviorální návrhové vzory.

2.8.1 Architektonické návrhové vzory

Architektonické návrhové vzory řeší problémy spojené s celkovým rámcem programu. Využívají se při vytváření programu od jeho základu, kdy slouží k tvorbě „kostry“ programu, na kterou budou následně navázány další komponenty. (Koseoglu, 2016)

Model-View-Controller

Využitím návrhového vzoru Model-View-Controller dochází zejména k oddělení programového kódu určeného k realizaci aplikační logiky a programového kódu určeného k implementaci uživatelského rozhraní. Využitím tohoto návrhového vzoru je aplikace rozdělena do tří částí, kterými jsou Model, View a Controller. (Koseoglu, 2016)

Model

Zabezpečuje aplikační logiku, neměl by obsahovat žádný programový kód, který má cokoliv společného s uživatelským rozhraním. V ABAPu tuto část představují třídy tvořené vývojářem. (Koseoglu, 2016)

View

Třídy, které realizují implementaci uživatelského rozhraní. V ABAPu je tato část zaštitována společností SAP. Jsou tedy k dispozici třídy, které je možné k tomuto účelu využít a není potřeba vytvářet třídy vlastní. (Koseoglu, 2016)

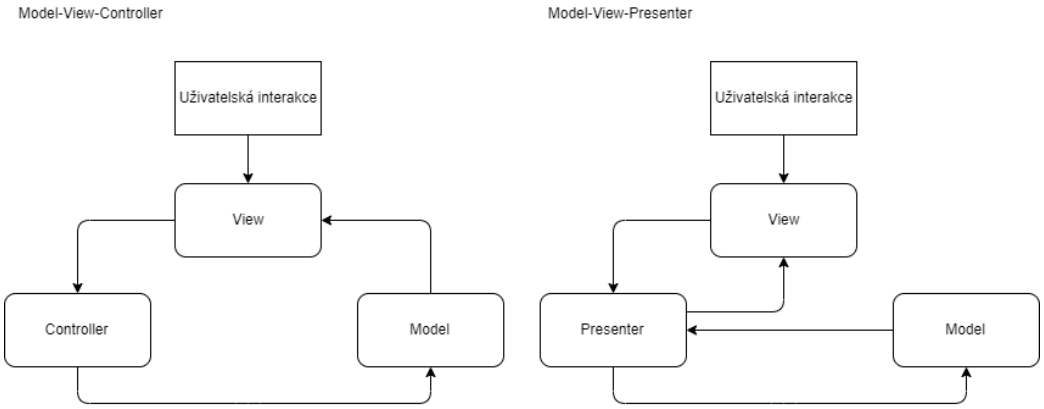
Controller

Komponenty Model a View jsou spolu propojeny prostřednictvím komponenty Controller. V ABAPu je nutno, aby komponentu Controller implementoval vývojář. (Hardy, 2019)

Model-View-Presenter

Návrhový vzor Model-View-Presenter je odvozený od návrhového vzoru Model-View-Controller. Základním rozdílem při nahrazení komponenty Controller komponentou Presenter je změna toku komunikace mezi jednotlivými komponentami. Komunikace mezi komponentami a interakcí ze strany uživatele je znázorněna na obrázku 2.6.

Obrázek 2.6 MVC a MVP



Zdroj: (www.medium.com) - vlastní zpracování

Při využití návrhového vzoru Model-View-Controller je interakce uživatele s komponentou View přeposlána komponentě Controller a následně je změněn Model, na základě kterého je generována komponenta View.

Při využití návrhového vzoru Model-View-Presenter dochází k oddělení vztahu mezi komponentami Model a View. Komunikační tok již není jednosměrný, ale dochází k obousměrné komunikaci mezi komponentami View-Presenter a Presenter-Model.

2.8.2 Tvořící návrhové vzory

Tvořící návrhové vzory se využívají při řízení tvorby instancí tříd, jelikož prosté vytvoření instance třídy např. s využitím unárního operátoru **`new`** nemusí být v některých situacích vhodné. (Koseoglu, 2016)

Návrhový vzor Singleton

Návrhový vzor Singleton řeší situaci, kdy má být v rámci běhu programu zabezpečeno, že neexistuje více než daný počet instancí vybrané třídy (standardně je pak požadováno, aby v rámci celého životního cyklu daného programu existovala právě jedna instance vybrané třídy). (Rupert, Wood, 2016)

Návrhový vzor Factory

Návrhový vzor Factory se využívá k abstrakci vytváření instance dané třídy. Vytváření instancí třídy může být často komplikované, například pokud je pro vytvoření instance potřeba zabezpečit specifikaci mnoha parametrů konstruktoru. Proto je vytvoření potřebné instance třídy zabezpečeno implementací tzv. factory, kterou je speciální třída abstrahující komplikace s vytvořením dané instance třídy. (Barbaric, 2009)

2.8.3 Strukturální návrhové vzory

Strukturální návrhové vzory se zabývají vztahy mezi instancemi tříd, tj. objekty, a slouží k tvorbě jednoduchých, účinných a flexibilních způsobů jejich interakce a spolupráce bez nutnosti tvorby tzv. pevných vazeb. (Koseoglu, 2016)

Návrhový vzor Proxy

Návrhový vzor Proxy se používá k obalení třídy jinou třídou. Rozhraní pro komunikace s původní třídou zůstává stejné, avšak aplikací tohoto návrhového vzoru může dojít ke změně chování v jejich jednotlivých metodách. (Koseoglu, 2016)

2.8.4 Behaviorální návrhové vzory

Behaviorální návrhové vzory řeší problematiku komunikace mezi objekty. Využívají se, pokud je nutné sdílet informace mezi objekty. (Koseoglu, 2016)

Návrhový vzor Strategy

Návrhový vzor Strategy je aplikován v situacích, kdy existuje více algoritmů k řešení daného problému, které mají stejné rozhraní. Program může následně za běhu rozhodnout, který z algoritmů by měl být v daném případě použit. (Barbaric, 2009)

2.9 Softwarové inženýrství

Pojem softwarové inženýrství vznikl v šedesátých letech dvacátého století, zejména v důsledku tzv. „softwarové krize“. (Sommerville, 2013)

Podstata této krize byla v tom, že většina programů měla ve svém výsledku nevalnou kvalitu, vyšší cenu, delší dobu vývoje a velmi ztíženou možnost údržby a rozšiřování o nové funkcionality. (Kadlec)

Mezi hlavní příčiny zmíněné krize patří zejména špatná komunikace, nesprávné odhady, nízká účinnost práce, podcenění hrozeb a rizik. (Kadlec)

Reakci na tuto krizi představuje softwarové inženýrství. Dochází ke vzniku prvotních více či méně propracovaných metodik vývoje softwaru, jejichž dodržování by mělo vést ke odstranění zmíněných příčin a důsledků softwarové krize. (Sommerville, 2013)

2.10 Metodiky vývoje software

Metodiky vývoje software je možné dělit na tradiční a agilní. Základní rozdíly mezi těmito metodikami jsou zejména v přístupu ke změnám, celkové dokumentaci, komunikaci se zákazníkem. (Kadlec, 2004)

2.10.1 Tradiční metodiky vývoje software

Tradiční metodiky vývoje software kladou velký důraz na přesné dodržování procesů, co nejlepší odhad jednotlivých termínů, detailní specifikace požadavků a jednoznačné stanovení rolí, kterých bývá mnoho a nepodílí se na jiných oblastech vývoje, než je definováno. (Myslín, 2016)

Velký důraz je kladen také na dokumentaci. Obsáhlá dokumentace se týká jak sběru požadavků, tak samotného vývoje, testování, předání i údržby. Zákazník tedy zpočátku projektu nezískává nic kromě rozsáhlé dokumentace. (Myslín, 2016)

Nutné je také zmínit obtížnost prosazování změn v projektu. Proces schvalování změnových požadavků může být vícestupňový a schválení může trvat i několik týdnů. Tyto změny opět vytvářejí nutnost jejich dokumentace a mohou celou fázi vývoje vrátit na začátek. (Myslín, 2016)

Mezi výhody těchto metodik je nutno uvést řád a předvídatelnost. Hlavní využití nacházejí tyto metodiky v projektech, v nichž je možno před jejich zahájením přesně definovat funkcionalitu, termíny a rozpočet. (Myslín, 2016)

2.10.2 Agilní metodiky vývoje software

Agilní metodiky vývoje software kladou důraz na jednotlivce a interakci místo přesného dodržování procesů. Reakce na případné změny je brána jako důležitější než dodržováním stanového plánu. Místo soustředění se na vyčerpávající dokumentaci je cílem fungující software. (Myslín, 2016)

Důležité je zmínit, že agilní přístup neznamená anarchii. Řád a pořádek v projektech je zachován a hlavní výhodou využití agilního přístupu je zefektivnění práce. Problémy při využití agilních metodik mohou také nastat zejména při jejich nesprávném

pochopení, případně při snaze o využití na nevhodných projektech. Pokud by byly všechny požadavky předem známy, obsáhlá dokumentace každého kroku vyžadována a prosazení změn obtížná, pak by snaha o využití agilních metodik mohla způsobit velké problémy. (Myslín, 2016)

Scrum

Scrum je rámec, s jehož pomocí mohou lidé řešit komplexní a adaptivní problémy a zároveň kreativně a produktivně dodávat produkty nejvyšší možné hodnoty. (www.scrum.org)

Scrum samotný je jednoduchý rámec pro efektivní týmovou spolupráci na komplexních produktech. Spoluautoři tohoto rámce Ken Schwaber a Jeff Sutherland vydali přehlednou monografii The Scrum Guide vysvětlující základní principy tohoto rámce, v níž jsou definovány události, role, artefakty a jejich vzájemné závislosti. (www.scrum.org)

První událostí, ke které dochází při zahájení projektu, je Kick-off meeting, neboli zahajovací meeting, na kterém dochází k seznámení jednotlivých zainteresovaných stran a společné diskusi o projektu. Následně dochází k události nazvané sprint, která je popsána níže. Po dokončení všech sprintů je projekt ukončen. (www.agile-mercurial.com)

Artefakty definované v metodice Scrum

Artefakty v metodice Scrum jsou Product Backlog, Sprint Backlog a Product Increment. (www.scrumalliance.org)

Product Backlog reprezentuje seznam veškerých požadavků, které mají být realizovány v rámci nadcházejících sprintů. (www.scrumalliance.org)

Sprint Backlog představuje seznam položek z Product Backlogu, které mají být v daném sprintu realizovány. (www.scrumalliance.org)

Product Increment je sumou všech položek Product Backlogu, které jsou v rámci daného sprintu dokončeny a všech Product Incrementů předcházejících sprintů. Product Increment pak představuje krok k dokončení projektu. (www.scrum.org)

Události definované v rámci metodiky Scrum

Hlavní událostí definované v rámci metodiky Scrum je tzv. sprint. Jedná se časově ohraničený interval, během kterého jsou realizovány zvolené uživatelské požadavky. (www.scrum.org)

Před zahájením každého sprintu je nutné naplánovat, které položky z Product Backlogu budou v jeho rámci realizovány. K tomu je určený meeting nazvaný Sprint Planning. (www.scrum.org)

Během daného sprintu dochází ke každodennímu meetingu všech členů Development Teamu nazvaného Daily Standup, na kterém si sdělí, na čem kdo pracuje a zdali nastal nějaký problém. (www.scrum.org)

Po ukončení daného sprintu probíhají meetingy typu Sprint Review a Sprint Retrospective. Na Sprint Review meetingu jsou Product Ownerovi prezentovány výstupy daného sprintu. Na Sprint Retrospective meetingu diskutují členové Development Teamu o tom, co během daného sprintu fungovalo správně, případně co by se mělo změnit. (www.scrum.org)

Role definované v metodice Scrum

Role v metodice Scrum jsou tři: Product Owner, Scrum Master a Development Team. (www.scrumalliance.org)

Product Owner je osoba, která reprezentuje zákazníka. Product Owner má znalosti o tom, co se od projektu očekává, zná funkcionální stránku projektu a dokáže ohodnotit přidanou hodnotu nových funkcionalit. (www.scrumalliance.org)

ScrumMaster je osoba, která řeší veškeré problémy, které by mohly vadit vývojářům v práci. Dohlíží na správný chod meetingů a správné užití metodiky Scrum. (www.scrumalliance.org)

Development Team je vývojářský tým složený nejen z vývojářů, ale i například z testerů a analytiků. Spolupracuje s Product Ownerem a je zodpovědný za plnění položek daného sprintu. (www.scrumalliance.org)

3 Analýza současného stavu

V následujících podkapitolách bude představena společnost a problém, který je v rámci této práce řešen. (www.new.siemens.com)

3.1.1 Představení společnosti

Historie společnosti Siemens začala roku 1847, kdy Werner von Siemens spolu s Johannem Georgem Halskem založili telegrafní stavební společnost s názvem Telegraphen-Bauanstalt von Siemens & Halske, ze které se postupně stala dnešní společnost Siemens. (www.new.siemens.com)

Společnost Siemens s. r. o. má v Česku oficiální zastoupení od roku 1890. Ještě před vznikem tohoto zastoupení v roce 1885 však dodala společnost komplexní osvětlení do Stavovského divadla v Praze. Po vzniku Československa v roce 1918 byly postaveny výrobní závody společnosti, ve kterých se vyráběly např. silnoproudá zařízení pro elektrárny, elektromotory, generátory. (www.new.siemens.com)

Roku 1945 došlo k znárodnění firmy a výrobních závodů. Roku 1971 byla otevřena technicko-poradenská kancelář a byly obnoveny dodávky moderních technologií. Roku 1990 pak došlo k plnému navrácení majetku společnosti a začala se rozrůstat skupina sdružující mnoho servisních, obchodních a výrobních závodů. (www.new.siemens.com)

3.1.2 Představení problému

Ve společnosti Siemens s. r. o. se v současné době využívá program, jehož hlavní náplní je na základě dané objednávky nebo materiálu provést její kompletní rozpad na jednotlivé součástky a následně rozpady jednotlivých součástek až na co nejnižší možnou úroveň.

Tento program poskytuje informace o množství jednotlivých součástek nutných k pokrytí dané objednávky nebo materiálu a jejich dostupné množství ve vybraných závodech společnosti Siemens v ČR.

Výběr je možno i různě upravovat například zvolením kusovníku, ve kterém je definována hierarchická struktura všech materiálů, případně je možné filtrovat materiály v objednávce, které nejsou vyráběny přímo v závodech společnosti Siemens s. r. o., ale jsou nakupovány od externích dodavatelů.

Program je využíván jak zaměstnanci společnosti Siemens s. r. o. v oblasti nákupu a prodeje, tak i pro zobrazování informací o objednávkách pro vedení společnosti.

Došlo k rozhodnutí, že stávající program není dostačující a je nutné jej optimalizovat, případně vytvořit program nový. Základní funkcionalita by měla zůstat stejná, a to rozpad materiálu nebo objednávky na součástky, avšak bližší specifikace a případy užití nejsou zcela definované a mohou se na základě zpětné vazby od koncových uživatelů měnit.

Hlavním problémy spolu s prvotními návrhy řešení jsou vypsány níže:

Architektura programu

Architektura stávajícího programu je takřka monolitická. Aplikační, prezentační i databázová vrstva jsou úzce spjaty a provázány, což znesnadňuje změny v kódu a hledání možných chyb.

Řešením by bylo využití návrhového vzoru Model-View-Prestenter, kterým by bylo zajištěno rozdělení programu do tří částí.

Čitelnost kódu

Ztížená čitelnost a pochopitelnost kódu, která je z velké části způsobena mnohými úpravami v procedurálně psaném kódu.

Jelikož programovací jazyk ABAP podporuje objektově orientovaný přístup, mělo by jeho správné využití vést k lepší pochopitelnosti a čitelnosti kódu.

Chybové stavy

Neřešení chybových stavů. Pokud v aplikaci dojde k chybovému stavu, není tento chybový stav žádným způsobem ošetřen. Uživateli tedy program ukončí svůj běh a zobrazí chybovou hlášku, která uživateli programu nebude dávat žádný smysl.

Chybové stavy je možno v programech psaných v jazyce ABAP odchytávat a řešit jak při použití procedurálního, tak objektově orientovaného způsobu programování.

Komunikace s databází

V programu je mnoho případů získávání dat z databáze, avšak často nevhodně řešených. Například se jedná o iterování seznamu a následné získávání dat z databáze pro každou položku zvlášť, což by se dalo řešit získáním všech potřebných dat pro všechny položky najednou.

Rychlost běhu programu

V případě zvolení více materiálů nebo objednávek je běh programu nedostatečně rychlý. Je to způsobeno již výše vypsanými problémy, kdy pravděpodobně nejproblematičtější oblastí bude řešení komunikace s databází.

4 Návrh a implementace aplikace k zobrazení rozpadu materiálů

V rámci této kapitoly je řešen návrh a implementace představeného problému. Součástí této kapitoly je také využití metodiky vývoje softwaru a její přizpůsobení dané situaci.

4.1 Volba, úprava a aplikování metodiky

Před začátkem práce na programu je nutné na základě analýzy současného stavu zvolit vhodnou metodiku vývoje softwaru.

Je potřeba si uvědomit, že vývoj bude prováděn pouze jednou osobou, proto není zcela možné aplikovat metodiky určené pro vývojové týmy o více lidech.

Jako vhodné se jeví vybrat vhodnou metodiku a tu upravit pro potřeby daného projektu, přičemž jednou z možností je také volbu metodiky vývoje software vůbec neřešit. Tento přístup je však dostatečný pouze pro opravdu malé projekty, kde by volba metodiky a její využití bylo téměř náročnější než samotný projekt.

Jelikož se požadavky koncových uživatelů mohou ještě měnit, není ze strany zadavatele důraz na obsáhlou dokumentaci každé aktivity na projektu a je možnost s koncovými uživateli komunikovat. Z těchto důvodů se jako vhodné řešení nabízí využití agilní metodiky.

Využití metodiky Scrum (www.scrum.org) se jeví jako správné řešení, ovšem je nutné vybrané elementy této metodiky upravit na míru řešeného projektu.

4.1.1 Úprava zvolené metodiky

Jelikož je zvolená metodika Scrum určena pro programátorské týmy o více členech, je nutné ji upravit pro potřeby daného projektu.

Mezi prvky, které při vývoji programového systému jednotlivcem nejsou potřeba, se řadí zejména velký počet pracovních schůzek, které mají pomáhat v kooperaci mezi vývojáři v týmu.

Mezi hlavní prvky, které by v rámci metodiky Scrum měly význam pro samotného vývojáře, pak patří:

Sprint

Časový interval mezi 1 až 6 týdny, během kterého jsou zvolené požadavky implementovány do projektu. Doba trvání všech sprintů by měla být stejná. (Šochová)

Product Backlog

Seznam, ve kterém jsou ukládány veškeré požadavky na aplikaci, které se v rámci projektu mohou měnit. (www.agilealliance.org)

Sprint Backlog

Seznam, ve kterém jsou uloženy veškeré požadavky na aplikaci, které budou řešeny v daném sprintu. Tento seznam by se po vytvoření neměl měnit, ale je to možné, pokud se prostřednictvím změn dosáhne snazšího splnění požadavků pro daný sprint. (www.visual-paradigm.com)

Developer (Development Team)

Samotný vývojář, který představuje vývojový tým. Zajišťuje také odhad náročnosti úkolů v Backlogu. (www.scrum.org)

Product Owner

Osoba reprezentující zákazníka, která zná funkcionální stránku projektu. Ví, co od aplikace požaduje a jaká je její přidaná hodnota. Tato osoba pak vytváří položky v Product Backlogu. (www.scrum.org)

Manager (Scrum Master)

Manažer je v této úpravě omezená role Scrum Mastera. Je to osoba zodpovědná za to, že vývojovému týmu nic nebrání v práci. (www.scrum.org)

Zahajovací meeting

Prvotní meeting, na kterém je představena vize projektu a obecně nastíněny požadavky. (www.agileconnection.com)

Sprint Review a Planning Meeting

Meeting, který se uskutečňuje po ukončení jednotlivého sprintu a na kterém je zákazníkovi předvedena implementace požadavků daného sprintu. Jelikož v případě práce jednoho vývojáře nebude implementováno tak velké množství funkcionalit jako je tomu v případě velkého vývojářského týmu, jeví se jako vhodné tento meeting sjednotit

s Planning meetingem, na kterém se plánuje, jaké požadavky budou implementovány v následujícím sprintu. (www.yodiz.com)

4.1.2 Příprava zvolené metodiky

Nejprve je nutné provést Zahajovací meeting, během kterého by mělo dojít ke zjištění vize projektu, sepsání základních úkolů do Product Backlogu, představení osob a jejich rolí.

Důležité je také zvolit délku sprintu. Jako ideální doba se jeví dva týdny. Jelikož se jedná o jednočlenný vývojový tým, kratší časový interval by znamenal nedostatek změn a delší časový interval by pak zvyšoval riziko, že nebudou požadavky dostatečně diskutovány a případně chybným způsobem implementovány.

Poté dojde ke Sprint Planning Meetingu, na kterém bude určeno, na jakých požadavcích se v rámci sprintu bude pracovat, a tyto budou zapsány do Spring Backlogu.

Po uběhnutí doby sprintu dojde k Sprint Review, kde budou Product Ownerovi prezentovány výsledky sprintu.

Následně budou všechny zmíněné kroky, od Sprint Planning Meetingu, opakovány, dokud nebudou veškeré požadavky v Product Backlogu úspěšně vyřešeny.

4.2 Zahajovací meeting

Upravenou metodiku je nyní nutné aplikovat na daný projekt.

Tento meeting už z části proběhl, jelikož základní vize a obecné zmínění požadavků byly nutné ve fázi analýzy současného stavu. Základní vize je tedy ve shrnutí optimalizace stávajícího řešení vytvořením řešení nového, ve kterém budou řešeny výtky zmíněné v části této práce obsahující analýzu současného stavu.

Projektový tým, jejich role a popis jsou vypsány v tabulce 4.1.

Tabulka 4.1 Projektový tým, jejich role a popis

Jméno a příjmení	Role	Popis role
Jan Hošna	Developer	Zajišťuje vývoj aplikace, určuje časovou náročnost úkolů v Backlogu.
Jan Dobiáš	Manager	Zajišťuje, že Development Teamu nic nebrání ve výkonu práce.
Karel Skypala	Product Owner	Představuje zákazníka, určuje vizi a položky v Backlogu, kterým přiřazuje prioritu.

Zdroj: vlastní zpracování

Na meetingu došlo k vytvoření šablony na jednotlivé položky Backlogu. Každá položka má své atributy, které jsou vypsány v tabulce 4.2.

Tabulka 4.2 Atributy a popis položek v Backlogu

Název atributu	Popis
Jedinečný identifikátor	Číslo, podle kterého lze identifikovat danou položku.
Typ	Typ položky může být buď nová funkcionalita (Feature), nalezená chyba (Defect), technická práce (Technical Work), nebo získání vědomosti (Knowledge Acquisition).
Téma	Oblast, která je řešena více položkami.
Status	Položka může být ve stavu k udělání (Todo), zpracovávání (In Work), nebo hotovém (Done).
Příběh, nebo popis	Buď se jedná o uživatelský příběh, ve kterém je popsána jeho role a jeho požadavek, nebo o popis prostým textem.
Časový odhad	Doba v hodinách, kterou by mělo trvat splnění položky.
Priorita	Priorita splnění položky stanová Product Ownerem.
Akceptační kritéria	Kritéria pro uznání splnění položky.
Poznámka	Doplňující informace k položce.

Zdroj: vlastní zpracování

Na zahajovacím meetingu došlo k definování položek typu nová funkcionalita (feature). Zejména se přitom jedná o funkcionality, které uživatelé využívají ve stávající verzi programu.

Byly definovány rovněž položky typu získání vědomostí (Knowledge Acquisition) a technická práce (Technical Work). Tyto položky souvisejí s nutností vytvoření základu programu a jeho architektury, na který budou později přidávány nové funkcionality. Z toho je zřejmé, že toto získání vědomostí a technická práce musí nastat před tím, než budou řešeny jednotlivé nové funkcionality (feature).

Problematika nutnosti vytvoření základní architektury programu a získání vědomostí, jakým způsobem k řešení problému přistoupit, jsou při využití agilní metodiky poměrně těžko uchopitelné, jelikož veškerá práce na projektu by měla zákazníkovi přinášet užitek, avšak správné vytvoření základní architektury pro zákazníka žádný přínos z hlediska přínosu jeho užitku nemá. Proto se tyto počáteční práce, které je nutné udělat, řadí do tzv. nultého sprintu. (Šochová, Kunce, 2019)

4.3 Nultý sprint – Implementace MVP vzoru v ABAP

Na zahajovacím meetingu došlo k domluvě na nultém sprintu, během kterého budou realizovány dvě položky zobrazené v tabulce 4.3.

Tabulka 4.3 Nultý sprint

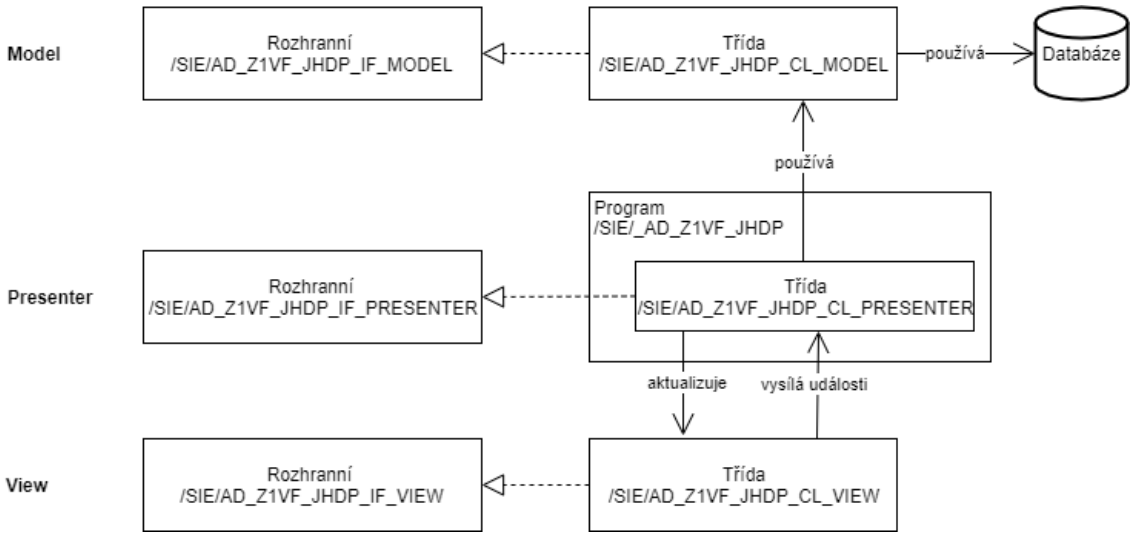
Položka z Backlogu	**Typ**	**Odhad doby trvání**
Zkoumání možností implementace návrhového vzoru MVP v jazyce ABAP	Získání vědomostí	24 hodin
Implementace základu návrhového vzoru MVP	Technická práce	16 hodin

Zdroj: vlastní zpracování

Je nutné vytvořit základní architekturu programu, na kterou budou postupně navázány nové funkcionality. Jako ideální se jeví využití návrhového vzoru Model-View-Presenter, jelikož může dojít ke změně požadavku na grafický výstup pro uživatele a tento návrhový vzor usnadňuje splnění tohoto požadavku.

Základ programu se tedy bude skládat ze tří základních částí znázorněných na obrázku 4.1. Každá z částí je v návrhu představována programovým rozhraním zejména z důvodu umožnění testování jednotlivých částí. Důležitou roli však hraje rozhraní pro část View, jelikož využité technologie pro grafické uživatelské prostředí uživatele se často mění. V tomto případě nebude nutné přepisovat celý program, ale bude stačit vytvořit novou třídu implementující dané rozhraní a využívající modernější způsoby a touto nově implementovanou třídou pak nahradit stávající třídu implementující komponentu View.

Obrázek 4.1 Základní části programu



Zdroj: vlastní zpracování

Model

Model je představován globální třídou **`/SIE/AD_Z1VF_JHDP_CL_MODEL`**, která působí jako centrální bod části Model. Součástí Modelu budou i lokální třídy, které budou zajišťovat určité funkcionality aplikace.

Jediným požadavkem v této fázi na třídu implementující komponentu Model je, aby tato třída dokázala vytvořit na základě čísla materiálu jeho úplný rozklad a vrátit jej v podobě interní tabulky.

Presenter

Komponenta Presenter je reprezentována spustitelným programem **`/SIE/AD_Z1VF_JHDP`** a třídou **`/SIE/AD_Z1VF_JHDP_CL_PRESENTER`**, která bude sloužit k zajištění komunikace mezi komponentami Model a View. V jazyce ABAP je však nutné v tomto programu definovat veškeré vstupní parametry, na základě kterých

je automaticky vygenerována obrazovka pro vstupní parametry. Proto je nutné, aby součástí této části byl kromě třídy i spustitelný program.

View

Komponenta View je implementována prostřednictvím třídy **`/SIE/AD_Z1VF_JHDP_CL_VIEW`**. V jazyce ABAP je část View řešena prostřednictvím tříd vytvořených společností SAP. Tyto třídy se však různí. Některé jsou tvořeny pro procedurální programy, jiné pro objektově orientované. Navíc nemají jednotné rozhraní, díky kterému by byly lehce obměnitelné.

4.3.1 Sprint Review nultého and Planning Meeting prvního sprintu

Během nultého sprintu došlo k implementaci návrhového vzoru Model-View-Presenter v programovacím jazyce ABAP. Tím došlo k vytvoření „kostry" programu, ke které budou přidávány nové funkcionality.

Na tomto meetingu došlo zejména k definování uživatelských příběhů, které byly přidány do Backlogu a následně určeny k implementaci během prvního sprintu.

4.4 První sprint – Implementace základních funkcionálních požadavků

V prvním sprintu byly implementovány základní funkcionální požadavky. Z Backlogu se jednalo o položky zobrazené v tabulce 4.4.

Tabulka 4.4 První sprint

Položka z Backlogu	Typ	Odhad doby trvání
Jako uživatel chci vědět, z čeho sestává daný materiál.	Funkcionalita	40 hodin

Zdroj: vlastní zpracování

Tento úkol je náročný zejména z nutnosti pochopení, jak rozklad materiálu v systému SAP funguje. K tomuto rozkladu se využívá funkční modul **`CS_BOM_EXPL_MAT_V2`**, který na základě parametrů rozloží daný materiál. Využití tohoto funkční modulu je také požadavkem společnosti Siemens s. r. o., jelikož se jedná o funkční model vyvinutý společností SAP. Bohužel pro tento funkční modul neexistuje oficiální dokumentace v českém, ani v anglickém jazyce. Je tedy nutné spoléhat na překlad z německé dokumentace, případně využít návody na Internetu.

Parametrů tohoto funkčního modulu je mnoho, avšak pro splnění požadovaných požadavků uživatelů bude, alespoň prozatím, stačit využít parametry vypsané v tabulce 4.5.

Tabulka 4.5 Parametry funkčního modulu CS_BOM_EXPL_MAT_V2

Název parametru	**Popis parametru**
MTNRV	Číslo materiálu, který se má rozpadnout.
CAPID	Kusovník, ve kterém je definováno, z čeho se materiály sestávají.
STLAL	Alternativa v kusovníku, která definuje, z čeho se daný materiál sestává.
MEHRS	Příznak, zdali se má materiál rozpadnout až na nejnižší možnou úroveň.
WERKS	Závod, podle kterého je určen kusovník.

Zdroj: vlastní zpracování

Výstupem tohoto funkčního modulu jsou primárně dvě tabulky. Jedna z nich obsahuje informace o materiálu, který byl rozpadnut, a druhá informace o položkách, na které byl daný materiál rozpadnut.

Důležité je také zmínit požadovaný výstup. Ten je zobrazen v tabulce 4.6. V této tabulce je rovněž znázorněno, které atributy je možné získat prostřednictvím zmíněného funkčního modelu, ostatní atributy pak bude nutné získat jiným způsobem.

Tabulka 4.6 Požadovaný výstup programu

Popis atributu	**Název atributu**	**Získání z funkčního modulu / zadán uživatelem**
Číslo materiálu	material_number	Ano
Jednotka	unit_of_measure	Ano
Množství	Quantity	Ano
Druh položky	item_category	Ano
Úroveň rozpadu	level_of_decomposition	Ano
Verze	document_version	Ano
Text CZ	text_czech	Ne
Text DE	text_german	Ne
Text EN	text_english	Ne
Třída ocenění	valuation_class	Ne
Druh pořízení	procurement_type	Ne
Druh zvláštního pořízení	special_procurement_type	Ne
Skupina nákupu	purchasing_group	Ne
Disponent	MRP_Controller	Ne
Výrobní dispečer	production_supervisor	Ne
Znak: Označení k výmazu materiálu na úrovni závodu	flag_material_for_deletion	Ne
Identifikátor pro materiál	identifier_of_material	Ne
Popis identifikátoru pro materiál	desc_of_identifier_of_material	Ne
Volně použitelná konsignační zásoba	valuated_unrestricted_use_stoc	Ne

(Celková zásoba) Oceněná, volně použitelná zásoba	unrestricted_use_consignment_s	Ne
Hmotnost 1 ks brutto	gross_weight	Ne
Celková hmotnost brutto	total_gross_weight	Ne

Zdroj: vlastní zpracování

Ostatní atributy je nutné získat z databáze příkazem SQL Select. Ten je zobrazen na obrázku 4.2.

Obrázek 4.2 Získání zbývajících atributů z databáze

```
SELECT    mara~matnr AS material_number,
          maktc~maktx AS czech_text,   "TYPE makt-maktx,
          maktd~maktx AS german_text,  "TYPE makt-maktx,
          makte~maktx AS english_text, "TYPE makt-maktx,
          mbew~bklas AS valuation_class,    "     TYPE mbew-bklas,
          marc~beskz AS procurement_type,    "     TYPE marc-beskz,
          marc~sobsl AS special_procurement_type,    "     TYPE marc-sobsl,
          marc~ekgrp AS purchasing_group,    "     TYPE marc-ekgrp,
          marc~dispo AS MRP_Controller,    "     TYPE marc-dispo,
          marc~fevor AS production_supervisor,   "      TYPE marc-fevor,
          marc~lvorm AS flag_material_for_deletion,    "     TYPE marc-lvorm,
          mara~ze_identifier AS identifier_of_material," TYPE mara-ze_identifier,
          /sie/ad_z1ne_tkz~text_id AS desc_of_identifier_of_material,   "     TYPE /sie/ad_z1ne_tkz-text_id,
         SUM( mard~labst ) AS valuated_unrestricted_use_stoc,  "     TYPE mard-labst,
         SUM( mard~klabs ) AS unrestricted_use_consignment_s,    "     TYPE mard-klabs,
          mara~brgew AS gross_weight,    "     TYPE mara-brgew,
          mara~gewei AS total_gross_weight    "     TYPE mara-gewei,
          FROM mara LEFT JOIN mbew ON mara~matnr = mbew~matnr AND mbew~bwkey = @lv_plant_for_decomposition AND mbew~bwtar = ' '
                    LEFT JOIN marc ON mara~matnr = marc~matnr AND marc~werks = @lv_plant_for_decomposition
                    LEFT JOIN mard ON mara~matnr = mard~matnr AND mard~werks = @lv_plant_for_stock_calc"different werks
                    LEFT JOIN makt AS maktc ON mara~matnr = maktc~matnr AND maktc~spras = 'C'
                    LEFT JOIN makt AS maktd ON mara~matnr = maktd~matnr AND maktd~spras = 'D'
                    LEFT JOIN makt AS makte ON mara~matnr = makte~matnr AND makte~spras = 'E'
                    LEFT JOIN /sie/ad_z1ne_tkz ON mara~ze_identifier = /sie/ad_z1ne_tkz~identifier AND /sie/ad_z1ne_tkz~sprsl = 'C'
          WHERE mara~matnr IN @lt_mat_numbers_range
          GROUP BY mara~matnr,mbew~bklas, marc~beskz,marc~sobsl, marc~ekgrp, marc~dispo,  marc~fevor,marc~lvorm,mara~ze_identifier,
          /sie/ad_z1ne_tkz~text_id, mara~brgew,mara~gewei, maktc~maktx,maktd~maktx,makte~maktx, maktc~spras, maktc~spras, maktd~spras
          INTO TABLE @lt_select_output.
```

Zdroj: vlastní zpracování

Po získání všech potřebných atributů je nutné zobrazit uživateli výstup. To je možné předáním interní tabulky obsahující veškeré záznamy třídě **`CL_SALV_TABLE`**, která se postará o zobrazení výstupu uživateli.

4.5 Druhý sprint – Rozšíření základních funkcionálních požadavků

Během tohoto sprintu bylo nutno rozšířit stávající funkcionalitu programu. Jedná se o položky zobrazené v tabulce 4.7.

Tabulka 4.7 Položky druhého sprintu

Položka z Backlogu	Typ	Odhad doby trvání
Jako uživatel chci nahrát seznam materiálů z MS Excelu.	Funkcionalita	5 hodin
Jako uživatel chci vidět z čeho sestává materiál z dané objednávky.	Funkcionalita	10 hodin
Jako uživatel chci nahrát seznam materiálů z objednávek uložených v MS Excelu.	Funkcionalita	5 hodin
Jako uživatel chci vidět ve výstupu sumarizovaný materiál.	Funkcionalita	10 hodin
Jako uživatel chci vyfiltrovat materiál získávaný externě.	Funkcionalita	5 hodin
Jako uživatel chci zvolit maximální úroveň rozpadu.	Funkcionalita	5 hodin

Zdroj: vlastní zpracování

Nejprve bylo tedy nutné vyřešit získání materiálu z objednávky. V každé objednávce jsou obsaženy položky, které mají své pořadí, proto pro získání materiálu je potřeba vědět i pořadí položky v objednávce.

Pomocí těchto informací je možné z dané položky získat, jaký materiál má být rozpadnut. V objednávce je také množství, které bylo objednáno, což znamená, že nyní je potřeba rozpadat množství obsažené v objednávce.

Jelikož je nyní několik způsobů, jak získat materiál, který má být následně rozpadnout, bylo v programu často využito kompozice. Tok programu je představen na obrázku 4.3.

Obrázek 4.3 Tok programu

```
METHOD /sie/ad_z1vf_jhdp_if_model~process_inputs.
  mo_configuration =  NEW /sie/ad_z1vf_jhdp_cl_config( io_inputs_from_selection ).
  mo_output = NEW /sie/ad_z1vf_jhdp_cl_output( io_inputs_from_selection->get_layout( ) ).
  mt_data_to_be_decomposed = mo_configuration->get_fetch_data_strategy( )->fetch_data_to_be_decomposed( mo_configuration ).
  rv_number_of_rows_to_be_decomp = lines( mt_data_to_be_decomposed ).
ENDMETHOD.
```

Zdroj: vlastní zpracování

Nejprve uživatel zadá vstupní data, na základě kterých je nastaven konfigurační objekt, který zabezpečí správný běh programu. Také dojde k inicializaci výstupního objektu, který na základě vstupních dat nastaví výstupní rozložení obrazovky pro uživatele. V této fázi dojde také ke zpracování vstupních dat, která představují zdroj pro materiál, který má být rozpadnut. Ten je následně uložen do datové položky třídy, aby nemusel být opětovně generován. Metoda vrací počet materiálů, které budou dekomponovány.

Uživatel následně potvrdí, že chce spustit dekompozici materiálů, případně má možnost vrátit se zpět na výběrovou obrazovku a upravit vstupy. Po potvrzení, že zadané materiály mají být rozpadnuty, dojde k provedení metody, která zabezpečuje rozpad materiálů a získání chybějících atributů. Postup je zobrazen na obrázku 4.4.

Obrázek 4.4 Metoda k rozpadu materiálů

```
METHOD /sie/ad_z1vf_jhdp_if_model~fetch_data.
  me->check_authority( ).
  me->decompose_data( ).
  me->generate_output( ).
  ro_output = mo_output.
ENDMETHOD.
```

Zdroj: vlastní zpracování

Nejprve je ověřeno, zdali má uživatel dostatečné oprávnění k získání dat, jelikož se jedná o interní data, ke kterým má přístup jen omezený počet osob. Následně dojde k dekompozici vstupních materiálů prostřednictvím již zmíněného funkčního modulu a poté ke generování výstupu, který obsahuje i dodatečné atributy získané z databáze.

Možnost nahrávání dat z MS Excelu je zabezpečena využitím funkčních modulů společnosti SAP. Je nutné ověřit vstupní data a zabezpečit možné chybové stavy při nenalezení záznamů. Pro filtraci záznamů a omezení jejich výběru je možné je na základě vstupních hodnot získaných od uživatele nevybírat, případně z výstupní tabulky odstranit.

4.6 Třetí sprint – Testování aplikace, ošetření chybových stavů, překlady

Třetí sprint obsahoval položky zobrazené v tabulce 4.8.

Tabulka 4.8 Položky třetího sprintu

Položka z Backlogu	Typ	Odhad doby trvání
Jako uživatel chci vidět veškeré texty v češtině.	Funkcionalita	5 hodin
Jako uživatel chci vidět veškeré texty v angličtině.	Funkcionalita	5 hodin
Jako uživatel chci vidět při přihlášení v německém jazyce veškeré texty alespoň anglicky.	Funkcionalita	5 hodin
Implementace ošetřování chybových stavů.	Funkcionalita	20 hodin
Manuální testování programu ke zjištění možných chybových stavů a kontrola správnosti výstupů programu.	Technická práce	5 hodin

Zdroj: vlastní zpracování

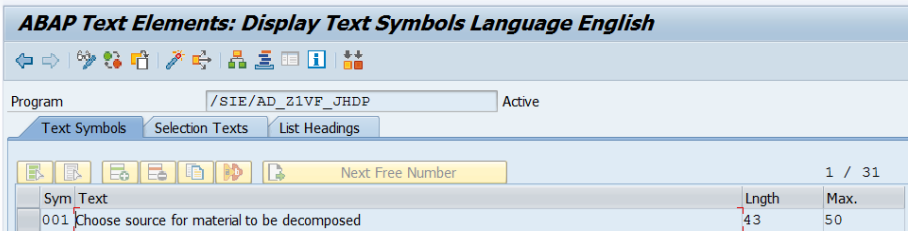
Překlady programů v systémech SAP jsou velmi důležité. Při přihlašování do systému SAP má uživatel možnost zvolit si jazyk přihlášení. To se stává poměrně problematickou záležitostí, jelikož mnoho programů v systému SAP bylo dříve psáno německy bez jakéhokoliv překladu. Nyní je snaha při komunikaci s uživatelem o prosazení anglického jazyka. To je problematika vývoje, ale koncového uživatele nezajímá, v jakém programovacím jazyce je psán kód, ale to, co vidí na obrazovce po spuštění programu.

To situaci ztěžuje o nutnost dodat překlady pro jazyk, který využívají koncoví uživatelé. V případě programu, který je v rámci této práce vyvíjen, se pak jedná o češtinu, avšak je nutné brát v potaz, že některý z uživatelů využívá jako jazyk pro přihlášení angličtinu nebo němčinu. V případě, že by nebyly k dispozici překlady, byly by uživateli zobrazeny technické názvy polí.

Veškeré texty v programu tedy musejí mít alternativy v českém, anglickém a německém jazyce. Tento problém byl vyřešen pomocí textových elementů. Tyto elementy jsou definovány v jazyce, v jakém je psán zdrojový kód programu, a následně jsou k nim vytvořeny překlady. Na základě jazyka přihlášení uživatele jsou pak zobrazeny dané překlady.

Příklad v anglickém jazyce je zobrazen na obrázku 4.5.

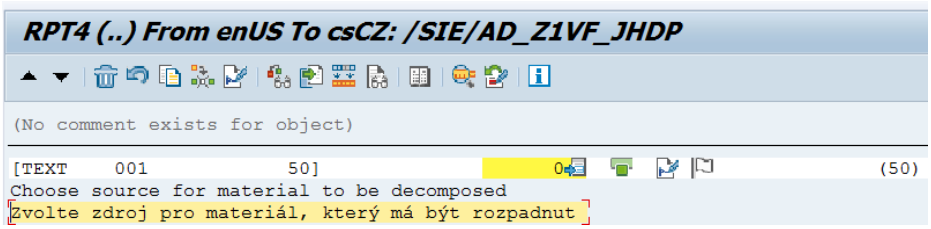
Obrázek 4.5 Překlad v anglickém jazyce



Zdroj: vlastní zpracování

Překlad do českého jazyka je zobrazen na obrázku 4.6.

Obrázek 4.6 Překlad do českého jazyka



Zdroj: vlastní zpracování

Překlady do německého jazyka jsou řešeny obdobným způsobem, avšak aby nedošlo k nesrovnalostem, je využito překladů anglických, jelikož aktuálně nikdo z uživatelů program v německém jazyce nevyužívá. Pokud by jej však někdo v tomto jazyce spustil, bylo by vhodné, aby se mu zobrazily alespoň anglické texty.

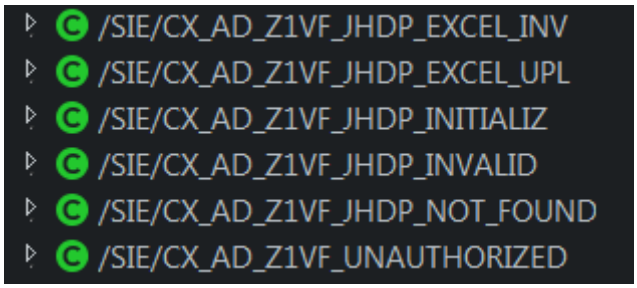
Výjimky a ošetření chybových stavů v programu je zcela nutné, jelikož uživatel získá informaci o tom, co v rámci běhu programu selhalo, a může například chybu vstupu sám napravit, případně může být snazší odhalit, že program nepracuje v některé části správně.

Pokud za běhu programu dojde k vyhození programové výjimky, je nutné ji zachytit. Pokud nedojde k zachycení programové výjimky, dojde k neočekávanému ukončení běhu programu a uživateli je zobrazen výpis problému, který

je velmi nápomocný pro hledání příčiny chyby v programu, avšak pro koncového uživatele je toto řešení pro svou technickou náročnost nevhodné.

Z těchto důvodu došlo k vytvoření několik základních výjimek, které mohou za běhu programu nastat. Výjimky jsou zobrazeny na obrázku 4.7.

Obrázek 4.7 Výjimky



Zdroj: vlastní zpracování

Jednotlivé výjimky popisují chybový stav, který v programu nastal. Jelikož názvy tříd nemohou být delší než 30 znaků, musely být názvy některých výjimek při dodržení jmenné konvence zkráceny.

V případě, že nastane v rámci běhu programu chybový stav, bude vyhozena programová výjimka. Tato programová výjimka obsahuje popis chybového stavu, případně je do popisu chybového stavu možno vložit jako argument např. číslo materiálu, pokud se jedná o vadný vstup.

Programové výjimky vzniklé v části Model zde nejsou zpravidla zachycovány, ale k jejich zachycení dojde až v části Presenter. Ze zachycených programových výjimek jsou získány informace o chybě a následné zobrazení programové výjimky uživateli provádí třída implementující rozhraní v rámci komponenty View. Odchyt výjimek je zobrazen na obrázku 4.8.

Obrázek 4.8 Odchyt výjimek

```
* Fetch data
    TRY.
        DATA(lv_number_of_rows_to_process) = mi_model->process_inputs( EXPORTING io_inputs_from_selection = io_inputs ).
        DATA(lv_should_proceed) = mi_model->show_pop_up_to_decide( lv_number_of_rows_to_process ).
        IF lv_should_proceed NE 'X'.
          mi_view->leave_to_list_processing( ).
          LEAVE LIST-PROCESSING.
        ELSE.
        ENDIF.
        DATA(lo_output_object) = mi_model->fetch_data( ).
      CATCH /sie/cx_ad_z1vf_jhdp_not_found /sie/cx_ad_z1vf_jhdp_initializ /sie/cx_ad_z1vf_jhdp_invalid /sie/cx_ad_z1vf_unauthorized INTO DATA(cx).
        mi_view->display_error( EXPORTING iv_text = cx->get_text( ) iv_type = 'E' ).
      CATCH /sie/cx_ad_z1vf_jhdp_excel_inv INTO DATA(cx_excel).
        mi_view->display_error( EXPORTING iv_text = cx_excel->get_text( ) iv_type = 'I' ).
        mi_view->display_error_excel_rows( CHANGING ct_error_excel_rows = cx_excel->mt_invalid_rows
                                                    ct_error_excel_rows_mat = cx_excel->mt_invalid_rows_mat ).
        exit.


    ENDTRY.
* Display data
    mi_view->display( lo_output_object ).
  ENDMETHOD.
```

Zdroj: vlastní zpracování

Programové výjimky mohou obsahovat více než jen text popisující chybu. Například v případě výjimky **`/SIE/CX_AD_Z1VF_JHDP_EXCEL_INV`**, která vzniká v případě, že uživatelem zadaná tabulka v MS Excelu obsahuje jak validní, tak nevalidní záznamy. Pokud by se uživateli zobrazila hláška, že tabulka v MS Excelu obsahuje nevalidní záznamy, tak v případě, že by zadal stovky řádků této tabulky, by se mu velmi těžce zjišťovalo, kde je chyba obsažena. Proto tato programová výjimka obsahuje tabulku nevalidních záznamů, které jsou po zobrazení chybové hlášky následně uživateli zobrazeny.

Jelikož došlo k dokončení vývoje programu, bylo nutné jej manuálně otestovat a ověřit s koncovými uživateli, že jeho funkcionality jsou správné.

4.7 Čtvrtý sprint – Analýza běhu programu a optimalizace běhu programu

V tomto sprintu byly řešeny položky vypsané v tabulce 4.9.

Tabulka 4.9 Položky čtvrtého sprintu

Položka z Backlogu	**Typ**	**Odhad doby trvání**
Analýza programu a testování rychlosti běhu programu.	Technická práce	20 hodin
Optimalizace programu.	Technická práce	20 hodin

Zdroj: vlastní zpracování

4.7.1 Analýza a testování rychlosti běhu programu

Nejprve bylo tedy nutné otestovat rychlost běhu programu a tento běh analyzovat. K tomu bylo vhodné využít nástrojů, které jsou vytvořeny společností SAP a k těmto účelům slouží.

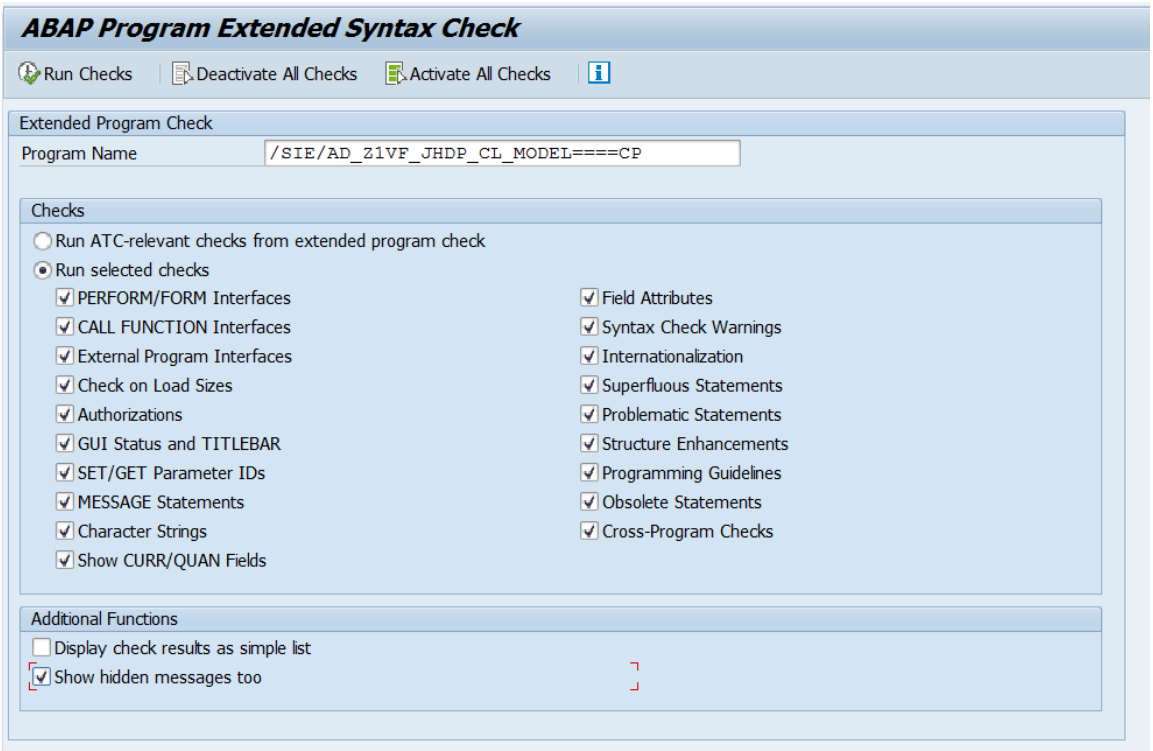
Code Inspector

Pomocí tohoto nástroje lze provést kontrolu zdrojového kódu. Tento nástroj je velmi prospěšný při vývoji k odhalení chyb, které mohou být snadno přehlédnuty, případně může například upozornit na využití zastaralých klíčových slov.

Nastavení testu je zobrazeno na obrázku 4.9 a výstup je zobrazen na obrázku 4.10. Jak je patrné, výtky vůči kódu programu jsou rozděleny do tří kategorií, a to Error,

Warning a Message. Error znamená, že se jedná o zásadní chybu, která může drasticky ohrozit běh programu. Warning znamená, že se nejedná o zásadní chybu, a reprezentuje např. využití zastaralých klíčových slov. V případě úpravy starších programů se mnohdy na použití zastaralých klíčových slov nehledí jako na problém, jelikož běh programu neohrožují. Message znamená informativní zprávu pro vývojáře. Nejedná se o problém, který by jakýmkoliv způsobem narušoval běh programu.

Obrázek 4.9 Code Inspector – vstup



Zdroj: vlastní zpracování

Obrázek 4.10 Code Inspector – výstup

SLIN Overview

Display Results Display All Results Display Single Test

Check for Program /SIE/AD_Z1VF_JHDP_CL_MODEL=	Error	Warnings	Messages
Test Environment	0	0	0
PERFORM/FORM Interfaces	0	0	0
CALL FUNCTION Interfaces	0	0	0
External Program Interfaces	0	0	0
Authorizations	0	0	0
GUI Status and TITLEBAR	0	0	0
SET/GET Parameter IDs	0	0	0
MESSAGE Statements	0	0	0
Character Strings	0	0	0
Show CURR/QUAN Fields	0	0	0
Field Attributes	0	0	0
Superfluous Statements	0	0	0
Syntax Check Warnings	0	0	0
Check on Load Sizes	0	0	0
Internationalization	0	0	0
Problematic Statements	0	0	0
Structure Enhancements	0	0	0
Programming Guidelines	0	0	0
Obsolete Statements	0	0	0
Cross-Program Checks	0	0	0
Hidden Errors and Warnings	0	0	0

Zdroj: vlastní zpracování

Mnohdy se stává, že kontrola programu pomocí nástroje Code Inspector zobrazuje falešné oznámení. Jelikož způsobů, jak v programovacím jazyce ABAP řešit nějaký problém, je mnoho, stává se občas, že Code Inspector vyhodnotí určitou oblast kódu jako vadnou, přestože tomu tak ve skutečnosti není.

Performance Trace/ Runtime Analysis a Single-transaction Analysis

Pomocí nástrojů Performance Trace/Runtime Analysis a Single-transaction Analysis je možno analyzovat běh programu. Jelikož nově vzniklý program má nahradit nyní používaný program, je vhodné je vůči sobě navzájem porovnat.

Výsledky běhu původního programu jsou zobrazeny v tabulce 4.10.

Tabulka 4.10 Výsledky běhu původního programu

Původní program				
Počet položek	1	10	100	1000
1. běh v sekundách	1,08825	3,128725	23,2042	205,681
2. běh v sekundách	0,990019	3,608811	23,34284	205,8586
3. běh v sekundách	0,943874	3,452254	23,28749	210,1307
4. běh v sekundách	0,897096	3,141701	22,90144	208,8295
5. běh v sekundách	0,847096	3,223319	23,09411	210,9192

Zdroj: vlastní zpracování

Výsledky běhu nového programu jsou zobrazeny v tabulce 4.11.

Tabulka 4.11 Výsledky běhu nového programu

Nový program				
Počet položek	1	10	100	1000
1. běh v sekundách	1,001193	2,867736	11,75759	114,1382
2. běh v sekundách	0,956227	2,641036	11,97722	113,6132
3. běh v sekundách	0,921629	2,195692	10,39396	109,2794
4. běh v sekundách	0,912632	2,105322	10,20661	108,2485
5. běh v sekundách	0,922851	2,195856	10,27249	110,2052

Zdroj: vlastní zpracování

Průměrné doby běhu obou programů jsou pro srovnání zobrazeny v tabulce 4.12.

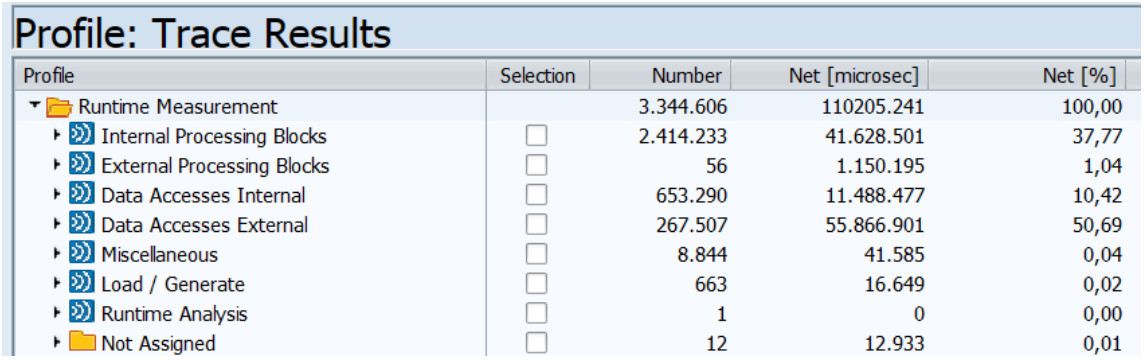
Tabulka 4.12 Průměrné doby běhu obou programů

Počet položek	1	10	100	1000
Průměrná doba běhu v sekundách – původní program	0,953267	3,310962	23,16601	208,1474
Průměrná doba běhu v sekundách – nový program	0,942906	2,401128	10,92157	111,0969

Zdroj: vlastní zpracování

Jak je z tabulky 4.12 patrné, nový program dokáže zpracovat stejné množství položek za menší časový interval, což byl jeden ze zadaných požadavků. Pomocí zmíněných nástrojů je možné analyzovat běh programu hlouběji než jen na základě výsledné velikosti časového intervalu nutného pro běh programu. Část z celkového výstupu je zobrazena na obrázku 4.11.

Obrázek 4.11 Část analýzy běhu programu



Zdroj: vlastní zpracování

Následně je možné zobrazit jednotlivé příkazy a jejich dobu trvání v tabulkovém zobrazení detailněji. Na obrázku 4.12 je zobrazena část výstupu seřazená od nejdelší doby trvání po nejkratší. Jedná se o poslední z měřených běhů nového programu při zpracování 1000 položek.

Obrázek 4.12 Část výstupu seřazená od nejdelší doby trvání po nejnižší

Hit List

Hits	Gross [microse...	Net [microse...	Gross [%]	Net [%]	Statement/Event
1	110.205.269	0	100,00	0	Runtime analysis
1	110.205.241	2.851	100,00	<0,01	Submit Report /SIE/AD_Z1VF_JHDP
2	110.202.390	393	100,00	<0,01	Program /SIE/AD_Z1VF_JHDP
6	109.321.870	125	99,20	<0,01	System Event <Generic Identifier>
1	109.319.129	75	99,20	<0,01	Dynpro Entry PROCESS DARK
2	109.315.737	15	99,19	<0,01	PAI Dynpro 1000
1	109.315.268	51	99,19	<0,01	Call M. /SIE/AD_Z1VF_JHDP_CL_PRESENTER=>MAIN
1	108.623.094	11	98,56	<0,01	Call M. {O:87*/SIE/AD_Z1VF_JHDP_CL_MODEL}->/SIE/AD_Z1VF_JHDP_IF_MODEL~FETCH_DA...
1	106.472.715	23.383	96,61	0,02	Call M. {O:87*/SIE/AD_Z1VF_JHDP_CL_MODEL}->DECOMPOSE_DATA
1.000	86.529.221	39.099	78,52	0,04	Call M. {O:89*/SIE/AD_Z1VF_JHDP_CL_CONFIG}->DECOMPOSE_DATA
1.000	86.414.535	209.008	78,41	0,19	Call Function CS_BOM_EXPL_MAT_V2
1.000	85.409.797	421.294	77,50	0,38	Call Function CS_BOM_EXPLOSION
26.792	60.504.949	74.364	54,90	0,07	Perform PFAD_PRUEFEN
8.446	41.897.396	475.698	38,02	0,43	Perform POS_PRUEFEN
26.792	40.433.126	225.612	36,69	0,20	Perform LTB_FUELLEN
21.631	39.478.661	361.850	35,82	0,33	Perform LTB_MAT_DATEN_UEBERNEHMEN
20.577	37.704.034	516.582	34,21	0,47	Call Function CS_ALT_SELECT_MAT
19.577	34.980.225	156.026	31,74	0,14	Perform STL_DATEN_HOLEN
9.405	21.163.657	129.681	19,20	0,12	Perform STPO_LESEN_DAT
9.405	20.993.005	387.072	19,05	0,35	Call Function GET_STPO
1.000	19.659.541	18.494.234	17,84	16,78	Call M. {O:89*/SIE/AD_Z1VF_JHDP_CL_CONFIG}->APPEND_TO_PROCESSING_TABLE
8.446	18.519.094	71.401	16,80	0,06	Perform POS_DATEN_HOLEN
23.861	16.331.631	102.355	14,82	0,09	Perform ALTSL_APPL_UNSPEZ
7.643	15.709.846	29.276	14,26	0,03	Perform KOPFDATEN_HOLEN
10.636	15.335.419	107.739	13,92	0,10	Perform STKO_LESEN
10.373	15.319.463	44.116	13,90	0,04	Perform D_KOPFDATEN_HOLEN
10.636	15.182.875	226.846	13,78	0,21	Call Function READ_STKO
959	15.098.411	150.024	13,70	0,14	Perform POS_PRUEFEN
20.577	14.742.946	326.160	13,38	0,30	Perform MAST_GET_FOR_SELID
20.577	14.328.721	116.368	13,00	0,11	Open SQL MAST
20.577	12.045.893	234.973	10,93	0,21	Fetch MAST
20.577	11.810.920	11.810.920	10,72	10,72	DB: Fetch MAST
21.009	11.458.034	170.004	10,40	0,15	Perform STZU_LESEN
21.009	11.288.030	1.303.500	10,24	1,18	Select Single STZU
9.405	10.340.581	273.390	9,38	0,25	Perform STPO_LESEN
9.405	10.067.191	54.615	9,13	0,05	Open SQL STPO
9.405	9.879.812	121.364	8,96	0,11	Perform MAT_PREREAD
21.009	9.637.820	9.637.820	8,75	8,75	DB: Open STZU
9.405	9.524.153	205.610	8,64	0,19	Call Function GET_STAS
9.405	8.806.938	182.634	7,99	0,17	Fetch STPO
9.405	8.624.304	8.624.304	7,83	7,83	DB: Fetch STPO
9.327	8.362.409	200.119	7,59	0,18	Call Function MATERIAL_PRE_READ_MC29S
10.636	7.345.127	109.446	6,66	0,10	Perform STKO_LESEN

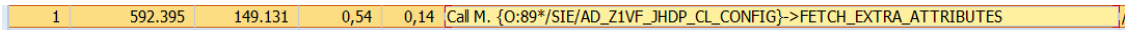
Zdroj: vlastní zpracování

Jak je vidět na obrázku 4.12, nejdelší časový interval potřebuje ke svému provedení metoda **`APPEND_TO_PROCESSING_TABLE`**, která po rozkladu materiálu zajišťuje jeho přidání k již rozpadlým materiálům. Tato metoda by měla být optimalizována, jelikož na základě měření bylo patrno, že při zvyšujícím se počtu položek k rozpadnutí dochází k násobné době potřebné k provedení metody. Aktuálně je doba trvání při rozpadu 1000 položek 18,5 sekund z celkových 110 sekund.

Dále je nutno zmínit funkční modul **`CS_BOM_EXPL_MAT_V2`**, který zajišťuje vlastní rozpad materiálu. Pokud je nutné provést rozpad násobně více materiálů, je pak nutné brát v potaz, že celková doba trvání běhu programu bude násobně delší. Na obrázku 4.12 je vidět, že z celkových 110 sekund trvá provádění tohoto funkčního modulu 86,4 sekundy.

Další důležitou částí programu je metoda **FETCH_EXTRA_ATTRIBUTES**, pomocí které dochází k získání dodatečných dat z databáze. Doba trvání metody je zobrazena na obrázku 4.13.

Obrázek 4.13 Doba trvání metody FETCH_EXTRA_ATTRIBUTES



Zdroj: vlastní zpracování

Doba provedení této metody trvá 0,6 sekund z celkových 110 sekund nutných pro běh programu. Tato doba trvání je přijatelná, jelikož dochází k získání mnoha potřebných dat.

Optimalizace problémových oblastí

Na základě provedených měření je vhodné pokusit se o snížení doby trvání problémových částí programu. Metoda **APPEND_TO_PROCESSING_TABLE** vytvářela zbytečně lokální proměnnou, do které byla při každé iteraci uložena celková oblast tabulky všech rozpadlých materiálů. To bylo vhodné při tvorbě programu, zejména pro kontrolu správnosti běhu programu. Avšak při tisíci položkách je nutné tento krok vynechat.

Optimalizace funkčního modelu **CS_BOM_EXPL_MAT_V2** je značně komplikovanější, jelikož se jedná o standartní funkční modul společnosti SAP a není tedy možné do něj bezproblémově zasahovat. Jako řešení se jeví pokusit se jej využívat jen pokud je to nezbytně nutné. Mnoho materiálů se totiž na nic nerozpadá, proto je vhodné zkontrolovat v databázi, které materiály se nemohou rozpadnout a pro tyto materiály neprovádět zmíněný funkční modul.

Testování nové optimalizované verze

Nejprve je nutno zmínit změny, které oproti předchozí testované verzi nastaly. Zejména se jedná o optimalizaci problémových částí programu a také o přidání nápovědy pro uživatele, kterou si má možnost zobrazit po spuštění programu.

V tabulce 4.13 jsou zobrazeny doby trvání běhu nově optimalizovaného programu.

Tabulka 4.13 Doby trvání běhu nově optimalizovaného programu

Optimalizovaný program				
Počet položek	1	10	100	1000
1. běh v sekundách	1,194931	2,025291	10,3657	92,26084
2. běh v sekundách	1,123238	2,031992	10,17491	91,4734
3. běh v sekundách	1,054654	2,113041	10,29375	93,15973
4. běh v sekundách	0,86574	2,08437	9,955646	93,06092
5. běh v sekundách	1,06744	2,17392	10,08904	92,84576

Zdroj: vlastní zpracování

Průměrné hodnoty časových intervalů nutných k běhu původního, nového i optimalizovaného nového programu jsou zobrazeny v tabulce 4.14.

Tabulka 4.14 Průměrné hodnoty časových intervalů nutných k běhu původního, nového i optimalizovaného nového programu

Počet položek	1	10	100	1000
Průměrná doba běhu v sekundách – původní program	0,953267	3,310962	23,16601	208,1474
Průměrná doba běhu v sekundách – nový program	0,9429064	2,401128	10,92157	111,0969
Průměrná doba běhu v sekundách – nový, optimalizovaný program	1,0612006	2,085723	10,17581	92,56013

Zdroj: vlastní zpracování

Z tabulky 4.14 je patrné, že běh programu při rozpadu jedné položky trvá průměrně déle, než tomu bylo v předchozí verzi. To je způsobeno přidáním uživatelské nápovědy, kterou systém načítá z databáze, a tato doba v měření způsobila prodloužení doby trvání. V případě rozpadu více položek trvá zmíněné získání uživatelské nápovědy velmi malou část doby trvání vůči době běhu celého programu.

Na obrázku 4.14 je zobrazena část výstupu posledního měření pro rozpad 1000 položek z nástroje Single-transaction Analysis.

Obrázek 4.14 Single-transaction Analysis – rozpad 1000 položek

Hit List

Hits	Gross [microse...	Net [microse...	Gross [%]	Net [%]	Statement/Event	Program Called
1	92.260.870	0	100,00	0	Runtime analysis	SAPLS_ABAP_TRACE_DATA
1	92.260.844	2.478	100,00	<0,01	Submit Report /SIE/AD_Z1VF_JHDP	SAPLS_ABAP_TRACE_DATA
2	92.258.366	433	100,00	<0,01	Program /SIE/AD_Z1VF_JHDP	SAPLS_ABAP_TRACE_DATA
6	91.141.855	147	98,79	<0,01	System Event <Generic Identifier>	/SIE/AD_Z1VF_JHDP
1	91.139.706	114	98,78	<0,01	Dynpro Entry PROCESS DARK	SAPMSSY0
2	91.138.681	16	98,78	<0,01	PAI Dynpro 1000	SAPMSSY0
1	91.138.040	46	98,78	<0,01	Call M. /SIE/AD_Z1VF_JHDP_CL_PRESENTER=>MAIN	/SIE/AD_Z1VF_JHDP
1	90.317.205	13	97,89	<0,01	Call M. {O:57*/SIE/AD_Z1VF_JHDP_CL_MODEL}->/SIE/AD_Z1VF_JHDP_IF_MODEL~FETCH_DATA	/SIE/AD_Z1VF_JHDP_CL_PRESENTERCP
1	88.102.928	18.348	95,49	0,02	Call M. {O:57*/SIE/AD_Z1VF_JHDP_CL_MODEL}->DECOMPOSE_DATA	/SIE/AD_Z1VF_JHDP_CL_MODEL====
1.000	87.756.063	32.439	95,12	0,04	Call M. {O:55*/SIE/AD_Z1VF_JHDP_CL_CONFIG}->DECOMPOSE_DATA	/SIE/AD_Z1VF_JHDP_CL_MODEL====
959	87.644.241	199.705	95,00	0,22	Call Function CS_BOM_EXPL_MAT_V2	/SIE/AD_Z1VF_JHDP_CL_CONFIG===CP

Zdroj: vlastní zpracování

Na obrázku 4.14 je patrné, že z celkové doby trvání běhu programu, která činí 92,2 sekundy, trvalo 87,6 sekund provádění funkčního modulu **CS_BOM_EXPL_MAT_V2**.

Zbylých přibližně 4,6 sekund trvalo provedení všech ostatních komponent programu. Metoda **APPEND_TO_PROCESSING_TABLE**, jejíž provedení v předchozí verzi zabralo přibližně 18,5 sekund, potřebovala při tomto běhu programu ke svému provedení časový interval přibližně 0,05 sekundy. Výstup z měření je zobrazen na obrázku 4.15.

Obrázek 4.15 Metoda APPEND_TO_PROCESSING_TABLE

Hit List

Hits	Gross [microse...	Net [microse...	Gross [%]	Net [%]	Statement/Event
1.000	49.987	49.987	0,05	0,05	Call M. {O:55*/SIE/AD_Z1VF_JHDP_CL_CONFIG}->APPEND_TO_PROCESSING_TABLE

Zdroj: vlastní zpracování

Z předchozích výstupů je patrné, že problematickou oblastí při velkém počtu položek k rozpadnutí byl funkční modul **CS_BOM_EXPL_MAT_V2** a bylo nutné zkrátit jeho dobu běhu, po kterou provádí příslušný rozpad materiálu.

Při testování rozpadu většího množství, než bylo 1000 položek materiálu, se pak projevil problém při získávání dat z databáze prostřednictvím SQL příkazu **SELECT**. Získávání dodatečných atributů z databáze je pak obsaženo v metodě **FETCH_EXTRA_ATTRIBUTES**.

Při rozpadu 1000 položek je doba potřebná k provedení metody 0,6 sekund. Výstup je zobrazen na obrázku 4.16.

Obrázek 4.16 Rozpad 1000 položek

Hits	Gross [microse...	Net [microse...	Gross [%]	Net [%]	Statement/Event
1	603.931	144.809	0,65	0,16	Call M. {O:55*/SIE/AD_Z1VF_JHDP_CL_CONFIG}->FETCH_EXTRA_ATTRIBUTES

Zdroj: vlastní zpracování

Při rozpadu 2000 položek je doba potřebná k provedení metody 20 sekund. Výstup je zobrazen na obrázku 4.17.

Obrázek 4.17 Rozpad 2000 položek

Hits	Gross [microse...	Net [microse...	Gross [%]	Net [%]	Statement/Event
1	20.056.319	20.783	4,75	<0,01	Call M. {O:89*/SIE/AD_Z1VF_JHDP_CL_CONFIG}->FETCH_EXTRA_ATTRIBUTES

Zdroj: vlastní zpracování

Při rozpadu 3500 položek je doba k provedení metody 101 sekund. Výstup je zobrazen na obrázku 4.18.

Obrázek 4.18 Rozpad 3500 položek

Hits	Gross [microsec]	Net [microse...	Gross [%]	Net [%]	Statement/Event
1	101.711.210	32.237	7,95	<0,01	Call M. {O:89*/SIE/AD_Z1VF_JHDP_CL_CONFIG}->FETCH_EXTRA_ATTRIBUTES

Zdroj: vlastní zpracování

Je nutné si uvědomit, že rozpad jedné položky může znamenat nutnost získat dodatečné atributy pro desítky položek, na které se zadaná položka rozpadla. Avšak i přesto je doba 20 sekund pro rozpad 2000 položek a 101 sekund pro rozpad 3500 položek neakceptovatelně dlouhá.

První myšlenkou při hledání příčiny této situace je komplexnost příkazu **`SELECT`**, který navíc propojuje více tabulek a některé údaje agreguje. Proto došlo k rozpadu provádění příkazu **`SELECT`** dle jednotlivých tabulek a výsledky těchto jednotlivých vyhledávání jsou pak ukládány do hashovaných tabulek. Následně jsou položky, ke kterým je nutno získat dodatečné atributy, procházeny a k nim hledány dodatečné atributy z hashovaných tabulek.

Využitím tohoto způsobu se doba při rozpadu 2000 položek zkrátila na 6,4 sekundy. Výstup je zobrazen na obrázku 4.19.

Obrázek 4.19 Rozpad 2000 položek

Hits	Gross [microse...	Net [microse...	Gross [%]	Net [%]	Statement/Event
1	6.438.336	57.047	1,91	0,02	Call M. {O:89*/SIE/AD_Z1VF_JHDP_CL_CONFIG}->FETCH_EXTRA_ATTRIBUTES

Zdroj: vlastní zpracování

Tato hodnota je již přijatelnější, ale přesto při úvaze, že 1000 položek trvalo provedení rozpadu 0,6 sekund, je hodnota pro 2000 položek ve výši 6,4 sekundy stále značně vysoká. Jako rychlejší způsob se jeví provést příkaz **`SELECT`** pro určité množství položek (např. 500) a tuto logiku pak implementovat do programu.

Po zavedení této úpravy jsou výsledky měření zobrazeny v tabulce 4.15.

Tabulka 4.15 Výsledky měření po úpravě

Počet položek	1	10	100	1000	2000	3500
Doba provedení metody `FETCH_EXTRA_ATTRIBUTES` v sekundách	0,004	0,011	0,042	0,89	1,38	1,79

Zdroj: vlastní zpracování

Z tabulky 4.15 je patrné, že došlo k prodloužení doby trvání běhu metody při 1000 položkách. Nejedná se však o takovou dobu trvání, která by značným způsobem ovlivnila běh celého programu. Dále je z tabulky patrno, že při zvyšujícím se počtu položek již nedochází k tak markantnímu prodloužení doby provádění metody.

Při spuštění programu a zadání rozpadu pro 3500 položek byla celková doba jeho běhu 20 minut a 8 sekund. Výstup je graficky zobrazen na obrázku 4.20. Z této doby trvá přibližně 10 sekund provádění veškerého kódu programu s výjimkou funkčního modulu **`CS_BOM_EXPL_MAT_V2`**. Proto bylo nutné vyřešit, jak zkrátit potřebnou dobu provádění tohoto funkčního modulu.

Obrázek 4.20 Rozpad 3500 položek

Hits	Gross [microse...	Net [microse...	Gross [%]	Net [%]	Statement/Event	Program Called	Calling Program
1	1.208.466.290	0	100,00	0	Runtime analysis	SAPLS_ABAP_TRACE_DATA	SAPLS_ABAP_TRACE_[
1	1.208.466.252	5.432	100,00	<0,01	Submit Report /SIE/AD_Z1VF_JHDP	SAPLS_ABAP_TRACE_DATA	SAPLS_ABAP_TRACE_[
2	1.208.460.820	431	100,00	<0,01	Program /SIE/AD_Z1VF_JHDP	SAPLS_ABAP_TRACE_DATA	SAPLS_ABAP_TRACE_[
6	1.205.682.441	138	99,77	<0,01	System Event <Generic Identifier>	/SIE/AD_Z1VF_JHDP	/SIE/AD_Z1VF_JHDP
1	1.205.681.152	1.406	99,77	<0,01	Dynpro Entry PROCESS DARK	SAPMSSY0	SAPMSSY0
2	1.205.678.439	32	99,77	<0,01	PAI Dynpro 1000	SAPMSSY0	SAPMSSY0
1	1.205.677.783	61	99,77	<0,01	Call M. /SIE/AD_Z1VF_JHDP_CL_PRESENTER...	/SIE/AD_Z1VF_JHDP	/SIE/AD_Z1VF_JHDP
1	1.203.585.327	19	99,60	<0,01	Call M. {O:87*/SIE/AD_Z1VF_JHDP_CL_MOD...	/SIE/AD_Z1VF_JHDP_CL_PRESENTERCP	/SIE/AD_Z1VF_JHDP_(
1	1.200.960.798	80.085	99,38	0,01	Call M. {O:87*/SIE/AD_Z1VF_JHDP_CL_MOD...	/SIE/AD_Z1VF_JHDP_CL_MODEL====...	/SIE/AD_Z1VF_JHDP_(
3.500	1.199.287.497	162.692	99,24	0,01	Call M. {O:89*/SIE/AD_Z1VF_JHDP_CL_CON...	/SIE/AD_Z1VF_JHDP_CL_MODEL====...	/SIE/AD_Z1VF_JHDP_(
3.345	1.198.485.768	2.181.516	99,17	0,18	Call Function CS_BOM_EXPL_MAT_V2	/SIE/AD_Z1VF_JHDP_CL_CONFIG===CP	/SIE/AD_Z1VF_JHDP_(

Zdroj: vlastní zpracování

4.8 Pátý sprint – Návrh a implementace paralelního zpracování dat pomocí asynchronně vzdáleně volaného funkčního modulu

V tomto sprintu byly řešeny položky z následující tabulky 4.16.

Tabulka 4.16 Položky pátého sprintu

Položka z Backlogu	Typ	Odhad doby trvání
Návrh paralelního zpracování dat pomocí dialogových procesů.	Technická práce	20 hodin
Implementace paralelního zpracování dat pomocí dialogových procesů.	Technická práce	20 hodin

Zdroj: vlastní zpracování

4.8.1 Paralelizace programu pomocí asynchronních RFC

Identifikace oblastí vhodných k paralelnímu zpracování

Před implementací paralelního zpracování dat je nutné analyzovat program, jelikož paralelní zpracování by nebylo možné, pokud je nutné jeho určité kroky provádět sekvenčně. Rovněž je nutné analyzovat provádění funkčního modulu **`CS_BOM_EXPL_MAT_V2`**, jelikož z výkonnostního hlediska se tento modul jeví jako zásadní problém.

Uvedený funkční modul je využit v metodě **`DECOMPOSE_DATA`**, kde nejprve dojde k využití funkčního modulu a rozpadu položky, následně jsou na základě filtrační strategie odstraněny nepotřebné položky, dále je výstupní tabulka obsahující stovky sloupců převedena do tabulky obsahující pouze relevantní sloupce a nakonec jsou řádky nově vzniklé tabulky navázány do procesní tabulky, do níž jsou postupně přidávány všechny rozpady položek.

Jako vhodné využití paralelního zpracování se jeví rozdělit položky, které se mají rozpadnout do skupin, a ty pak podrobit paralelnímu zpracování. Následně je nutné výstupy jednotlivých paralelních procesů sjednotit do jediné procesní tabulky.

Velkou výhodou při návrhu paralelizace programu je to, že v metodě **`DECOMPOSE_DATA`** dochází pouze ke čtení dat a v rámci této metody nedochází k jakýmkoliv vzájemným závislostem, které by mohly dělat potíže. Problémy spojené se čtením a zápisem není nutno z programátorského pohledu řešit, jelikož je tato problematika zabezpečena systémem SAP.

Implementace

Při implementaci paralelního zpracování prostřednictvím asynchronně vzdáleně prováděného funkčního modulu je využito postupu popsaného v teoretické části této práce.

Nejprve je proveden funkční modul **SPBT_INITIALIZE** pro účely získání volných pracovních procesů ve skupině aplikačních serverů. Provedení funkčního modulu je zobrazeno na obrázku 4.21.

Obrázek 4.21 Provedení funkčního modulu SPBT_INITIALIZE

```
METHOD initialize_server_group.
  CALL FUNCTION 'SPBT_INITIALIZE'
    EXPORTING
      group_name                    = iv_server_group_name
    IMPORTING
      free_pbt_wps                  = rv_available_work_processes
    EXCEPTIONS
      invalid_group_name            = 1
      internal_error                = 2
      pbt_env_already_initialized   = 3
      currently_no_resources_avail  = 4
      no_pbt_resources_found        = 5
      cant_init_different_pbt_groups = 6
      OTHERS                        = 7.
  IF sy-subrc <> 0.
    RAISE EXCEPTION TYPE /sie/cx_ad_z1vf_jhdp_spbt_init
      EXPORTING
        msgid = sy-msgid
        msgty = sy-msgty
        msgno = sy-msgno
        msgv1 = sy-msgv1
        msgv2 = sy-msgv2
        msgv3 = sy-msgv3
        msgv4 = sy-msgv4.
  ENDIF.
ENDMETHOD.
```

Zdroj: vlastní zpracování

Poté je nutné zjistit počet dialogových procesů, které daný uživatel ještě může v systému využívat. K tomu je možno využít funkční modul **TH_USER_INFO**, provedení tohoto modulu je zobrazeno na obrázku 4.22.

Obrázek 4.22 Provedení funkčního modulu TH_USER_INFO

```
METHOD calculate_available_sessions.
  DATA lv_active_sessions TYPE i.
  DATA lv_max_sessions TYPE i.

  CALL FUNCTION 'TH_USER_INFO'
    EXPORTING
      client       = sy-mandt
      user         = sy-uname
    IMPORTING
      act_sessions = lv_active_sessions
      max_sessions = lv_max_sessions.

  rv_available_sessions_for_user = lv_max_sessions - lv_active_sessions.
ENDMETHOD.
```

Zdroj: vlastní zpracování

Po zjištění, která z těchto hodnot je menší, je možné rozdělit data do pracovních balíčků. Rozdělení do pracovních balíčků je zobrazeno na obrázku 4.23.

Obrázek 4.23 Rozdělení do pracovních balíčků

```
METHOD split_data_into_packages.
  DATA(index_where_to_split) = calc_split_index( ).
  DATA lt_data_package TYPE /sie/ad_zlvf_jhdp_if_fetch_d_s=>tt_data_to_be_decomposed.

  LOOP AT mt_data ASSIGNING FIELD-SYMBOL(<ls_data_row>).
    APPEND <ls_data_row> TO lt_data_package.
    IF sy-tabix MOD index_where_to_split EQ 0.
      mo_list_of_work_packages->add_item( lt_data_package ).
      clear lt_data_package.
    ENDIF.
  ENDLOOP.

  if lt_data_package is not initial.
    mo_list_of_work_packages->add_item( lt_data_package ).
  endif.

ENDMETHOD.
```

Zdroj: vlastní zpracování

Nyní je již možné provést asynchronně vzdáleně volaný funkční modul s pracovním balíčkem, který je třeba zpracovat. Také je potřeba definovat metodu pro získání dat z daného provedení asynchronně vzdáleně volaného funkčního modulu.

Do provedení asynchronně vzdáleně volaného funkčního modulu je nutno zaslat serializovaný konfigurační objekt, který obsahuje informace o nastavení rozpadu. Serializace je realizována provedením příkazu **`CALL TRANSFORMATION`** a z tohoto důvodu je nutno zabezpečit, aby serializovaná třída implementovala programové rozhraní **`IF_SERIALIZABLE_OBJECT`**. Serializace konfiguračního objektu je zobrazena na obrázku 4.24.

Obrázek 4.24 Serializace konfiguračního objektu

```
METHOD process_data_in_paralell.
  DATA lv_serialized_config TYPE string.
  DATA msg(80) VALUE space.

  CALL TRANSFORMATION id_indent SOURCE io_configuration = io_configuration RESULT XML lv_serialized_config.
```

Zdroj: vlastní zpracování

Dokud nejsou všechny balíčky přijaty, je nutno na jejich přijetí v rámci provádění programu čekat. V případě, že by došlo k nepředpokládanému problému, bude běh programu dle systémové proměnné, která limituje maximální dobu běhu programu, ukončen.

Při provádění asynchronně vzdáleně volaného funkčního modulu je nutno dále kontrolovat, zdali došlo k jeho úspěšnému provedení, případně se pokusit vyřešit nastalý problém a provedení asynchronně vzdáleně volaného funkčního modulu opakovat. Také je nutno definovat metodu, ve které dojde k získání dat vygenerovaných asynchronně

vzdáleně volaným funkčním modulem a definovat identifikátor pro název daného volání, podle kterého může být při návratu opět rozpoznán.

Po návratu z provedení asynchronně vzdáleně volaného funkčního modulu je navrácený balíček vložen do interní tabulky, která je seřazena dle názvu provedení asynchronně vzdáleně volaného funkčního modulu. Proto po návratu ze všech provedení asynchronně vzdáleně volaného funkčního modulu jsou výstupy seřazeny ve stejném pořadí, v jakém byly jednotlivé balíčky odesílány. Vložení navráceného balíčku do interní tabulky je zobrazeno na obrázku 4.25.

Obrázek 4.25 Vložení navráceného balíčku do interní tabulky

```
METHOD collect_data.
  DATA lt_enhanced_decomposed_data TYPE /sie/ad_z1vf_jhdp_tt_enh_data.
  DATA ls_package TYPE /sie/ad_z1vf_jhdp_tt_enh_data.

  RECEIVE RESULTS FROM FUNCTION '/SIE/AD_Z1VF_JHDP_FM_DEC_DATA'
   IMPORTING rt_decomposed_enhanced_data = lt_enhanced_decomposed_data
   EXCEPTIONS
     communication_failure = 1
     system_failure        = 2.

  mo_processed_work_packages->add_returned_item( iv_name = '' && p_task it_package_data = lt_enhanced_decomposed_data   ).
  mv_received_jobs = mv_received_jobs + 1.

ENDMETHOD.
```

Zdroj: vlastní zpracování

Po úspěšném ukončení všech provedení asynchronně vzdáleně volaného funkčního modulu je následné zpracování dat opět sekvenční. Srovnání běhu optimalizovaného a paralelizovaného programu je zobrazeno v tabulce 4.17.

Tabulka 4.17 Srovnání běhu optimalizovaného a paralelizovaného programu

Počet položek	1	10	100	1000	2000
Průměrná doba běhu v sekundách – původní program	0,953267	3,310962	23,16601	208,1474	532,581446
Průměrná doba běhu v sekundách – Optimalizovaný program	1,0612006	2,085723	10,17581	92,56013	369,468284
Průměrná doba běhu v sekundách – Paralelizovaný program	1,566937	2,686701	5,287489	25,972978	43,936864

Zdroj: vlastní zpracování

4.9 Šestý sprint – Paralelní zpracování dat prostřednictvím procesů na pozadí

V tomto sprintu byly řešeny položky zobrazeny v tabulce 4.18.

Tabulka 4.18 Položky šestého sprintu

Položka z Backlogu	Typ	Odhad doby trvání
Návrh paralelního zpracování dat pomocí procesů na pozadí.	Technická práce	20 hodin
Implementace paralelního zpracování pomocí procesů na pozadí.	Technická práce	20 hodin

Zdroj: vlastní zpracování

4.9.1 Paralelizace programu pomocí procesů na pozadí

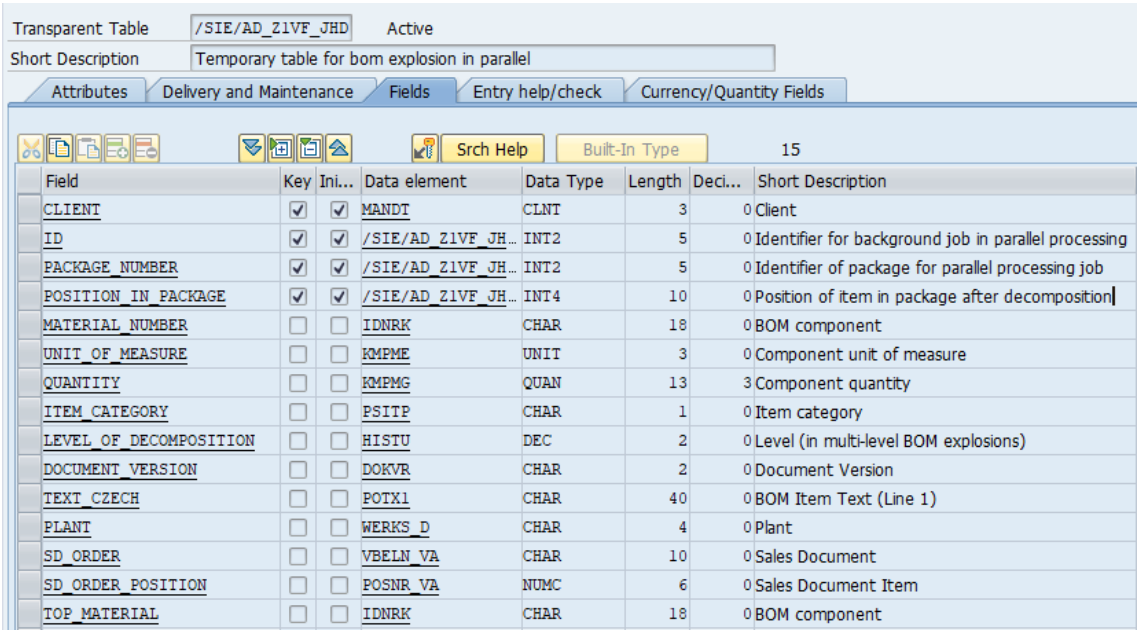
Identifikace částí programu vhodných k paralelnímu zpracování

Část programu vhodná k paralelnímu zpracování bude stejná, jako v případě paralelního zpracování pomocí dialogových procesů, a to provedení funkčního modulu **`CS_BOM_EXPL_MAT_V2`**.

Hlavním rozdílem při zpracování pomocí procesů na pozadí bude následné sjednocení paralelních běhů, jelikož při využití dialogových procesů je možno tuto situaci vyřešit nastavením metody nebo procedury, která se po dokončení paralelního běhu provede. To bohužel v případě paralelního zpracování pomocí procesů na pozadí nelze. Proto je nutné tuto problematiku v programu řešit vlastním programem.

Možností, jak řešit tuto problematiku, je přitom více. Jako ideální řešení se může jevit uložení dat do pracovní paměti, avšak při využití procesů na pozadí může dojít k využití procesu na pozadí běžícího na jiném aplikačním serveru, a tím může dojít k problémům. Jako vhodné řešení se jeví využití databázové tabulky, která bude sloužit k dočasnému ukládání dat za běhu programu, a po dokončení paralelních běhů bude obsah odpovídající danému běhu programu z tabulky vymazán. Tabulka je zobrazena na obrázku 4.26

Obrázek 4.26 Databázová tabulka k dočasnému ukládání dat



Transparent Table /SIE/AD_Z1VF_JHD Active

Short Description Temporary table for bom explosion in parallel

Attributes | Delivery and Maintenance | Fields | Entry help/check | Currency/Quantity Fields

Srch Help | Built-In Type | 15

Field	Key	Ini...	Data element	Data Type	Length	Deci...	Short Description
CLIENT	☑	☑	MANDT	CLNT	3	0	Client
ID	☑	☑	/SIE/AD_Z1VF_JH...	INT2	5	0	Identifier for background job in parallel processing
PACKAGE_NUMBER	☑	☑	/SIE/AD_Z1VF_JH...	INT2	5	0	Identifier of package for parallel processing job
POSITION_IN_PACKAGE	☑	☑	/SIE/AD_Z1VF_JH...	INT4	10	0	Position of item in package after decomposition
MATERIAL_NUMBER	☐	☐	IDNRK	CHAR	18	0	BOM component
UNIT_OF_MEASURE	☐	☐	KMPME	UNIT	3	0	Component unit of measure
QUANTITY	☐	☐	KMPMG	QUAN	13	3	Component quantity
ITEM_CATEGORY	☐	☐	PSITP	CHAR	1	0	Item category
LEVEL_OF_DECOMPOSITION	☐	☐	HISTU	DEC	2	0	Level (in multi-level BOM explosions)
DOCUMENT_VERSION	☐	☐	DOKVR	CHAR	2	0	Document Version
TEXT_CZECH	☐	☐	POTX1	CHAR	40	0	BOM Item Text (Line 1)
PLANT	☐	☐	WERKS_D	CHAR	4	0	Plant
SD_ORDER	☐	☐	VBELN_VA	CHAR	10	0	Sales Document
SD_ORDER_POSITION	☐	☐	POSNR_VA	NUMC	6	0	Sales Document Item
TOP_MATERIAL	☐	☐	IDNRK	CHAR	18	0	BOM component

Zdroj: vlastní zpracování

Na obrázku 4.26 je zobrazena tabulka, která je využita k dočasnému uchování dat při paralelním zpracování dat pomocí procesů na pozadí. Tabulka obsahuje klíčová pole, která slouží k jedinečné identifikaci běhu programu, pracovního balíčku a pozici materiálu po dekompozici.

Implementace

Při implementaci paralelního zpracování prostřednictvím procesů na pozadí je využito postupu popsaného v teoretické části této práce.

Tento postup musí být však značně upraven, jelikož při obvyklém využití procesů na pozadí nedochází k navrácení k programu, který vyvolal spuštění jiného programu prostřednictvím procesu na pozadí.

Nejprve je nutno rozhodnout o počtu pracovních balíčků, do kterých budou paralelně zpracovávaná data rozdělena. V případě procesů na pozadí je vhodnější zvolit počet manuálně, neboť v případě dynamického určování by mohlo dojít k problémům, jelikož uživatelé mohou být často neomezeni v počtu procesů na pozadí, které běží pod jejich účty, a tím způsobit zpoždění všech ostatních programů naplánovaných k zpracování pomocí procesů na pozadí.

K rozdělení do pracovních balíčků je využité stejné metody **SPLIT_DATA_INTO_PACKAGES**, jako tomu bylo v případě využití dialogových procesů.

Pro každý pracovní balíček byl vytvořen unikátní název, pomocí kterého jej bylo možno následně identifikovat, aby byla zaručeno stejné řazení zadaných dat jako v případě sekvenčního zpracování.

To je zabezpečeno získáním nejvyššího identifikátoru běhu programu a k němu připojení čísla balíčku, které je po každém naplánování zpracování prostřednictvím procesu na pozadí zvýšeno o jedničku.

Systém SAP zabezpečuje, že nedojde k problému při pokusu o čtení této tabulky více programy zároveň. Problém nenastane ani v případě mazání záznamů a současného čtení, jelikož je tato problematika řešena zámky na tabulce při čtení a mazání.

Problém, který je nutno ošetřit a který může nastat, je ten, že po získání nejvyšší hodnoty identifikátoru z tabulky nedojde k zarezervování tohoto identifikátoru v tabulce a každý další běh programu by získal stejné identifikační číslo. To by následně vedlo k problému při zápisu dat, jelikož by mohlo dojít k zápisu dat do tabulky, které již odpovídá jiný primární klíč, a tedy následně chybě zápisu. Tento problém lze řešit zamykáním tabulky při získávání klíče, který je následně v tabulce uložen, a poté je tabulka odemknuta.

Následně je možné provedením funkčního modulu **JOB_OPEN** nastavit název procesu na pozadí, který je možno tímto způsobem identifikovat. Tomuto procesu na pozadí je pak následně nastaven program s argumenty, který má být jeho prostřednictvím vykonán. Posledním krokem je provedení funkčního modulu **JOB_CLOSE**, kterému je zadáno, kdy má být proces na pozadí spuštěn.

Do programu, který je procesem na pozadí prováděn, je zaslán serializovaný konfigurační objekt stejně jako v případě využití asynchronně vzdáleně volaného funkčního modulu.

Následně je nutné počkat, až proběhne zpracování všech pracovních balíčků. To lze detekovat tak, že počet pracovních balíčků zpracovaných procesy na pozadí je stejný jako počet jedinečných záznamů pracovních balíčků s daným identifikátorem.

Po dokončení všech procesů na pozadí je možno získat zpracovaná data a seřadit je do stejného pořadí, jako by byla zpracována sekvenčně. Získání dat ze všech procesů je zobrazeno na obrázku 4.27.

Obrázek 4.27 Získání dat ze všech procesů

```
SELECT *
  FROM /sie/ad_z1vf_jhd INTO CORRESPONDING FIELDS OF TABLE rt_output_table
  WHERE id = lv_max_id
  ORDER BY package_number ASCENDING position_in_package ASCENDING.
```

Zdroj: vlastní zpracování

Následné zpracování dat je stejně jako v případě jejich zpracování pomocí asynchronní vzdálené volaní funkčního modulu sekvenční. Po dokončení zpracování dat nebo při detekci chyby je nutno smazat data, která byla vložena do procesní databázové tabulky. Srovnání běhu optimalizovaného a paralelizovaného programu je zobrazeno v tabulce 4.19.

Tabulka 4.19 Srovnání běhu optimalizovaného a paralelizovaného programu

Počet položek	1	10	100	1000	2000
Průměrná doba běhu v sekundách – původní program	0,953267	3,310962	23,16601	208,1474	532,581446
Průměrná doba běhu v sekundách – asynchronní RFC	1,566937	2,686701	5,287489	25,972978	43,936864
Průměrná doba běhu v sekundách – procesy na pozadí	1,954527	4,180688	6,978914	46,606346	139,972868

Zdroj: vlastní zpracování

Při testování bylo využito pěti procesů na pozadí. Z tabulky 4.19 je patrné, že využití paralelního zpracování prostřednictvím procesů na pozadí znatelně urychlilo běh programu, avšak proti využití dialogového paralelního zpracování nedosahuje lepších hodnot. Je to způsobeno zejména nutností zápisu a čtení do a z databázové tabulky.

4.10 Shrnutí a srovnání nového a původního programu

Z výstupů z předchozích podkapitol je patrné, že nejlepšího výkonu bylo dosaženo při využití paralelního zpracování asynchronně vzdáleně volaného funkčního

modulu. Následně při paralelním zpracování běží příslušné procesy na pozadí. Využití paralelního zpracování v obou případech vedlo k snížení potřebného výpočetního času.

Optimalizace zdrojového kódu a komunikace s databází velkým dílem přispěly k snížení potřebného času provádění programu.

Kromě zvýšení rychlosti běhu programu došlo také k přepsání původního procedurálního kódu do objektově orientovaného. Nové úpravy zejména v oblasti celkové architektury kódu budou v budoucnu méně náročné a bude snazší detekovat chyby v programu a předejít tím možným škodám.

Řešení chybových stavů a zlepšení čitelnosti kódu také pomohou zejména v dlouhodobém hledisku, jelikož noví uživatelé budou moci program užívat bez nutnosti čtení dlouhého návodu, ale v případě vzniku chyby budou vědět, k čemu při běhu programu došlo. Čitelnost kódu pomůže při rozšiřování programu o nové funkcionality, případně při hledání chyb.

5 Závěr

V diplomové práci byly v druhé kapitole vysvětleny pojmy a metodická východiska, která jsou potřebná k vývoji programů pro platformu SAP HANA v programovacím jazyce ABAP a také k paralelnímu zpracování dat těchto programů. Kapitola obsahuje informace o historii systému SAP a programovacího jazyka ABAP, jsou zde popsány jednotlivé verze systémů a jejich vlastnosti, architektura systému SAP a jeho vývoj v čase, jednotlivé moduly a základní typy aplikací v systému SAP. Jsou zde zmíněny i základní vlastnosti platformy SAP HANA, návrhové vzory a metodiky vývoje softwaru.

Problém, který byl v rámci práce řešen, je představen ve třetí kapitole včetně popisu současné situace při jeho řešení. V této kapitole je rovněž popsána společnost Siemens s. r. o. včetně její historie.

Kapitola čtvrtá obsahuje volbu metodiky pro vývoj softwaru, podle níž byly řešeny uživatelské požadavky. V kapitole je rovněž popsána implementace zadaných funkcionalit, nástrojů pro testování programu a řešení problematiky cizojazyčných překladů. Výsledný program byl nejprve navržen, implementován a po dokončení jeho implementace dále optimalizován.

Následně pak bylo aplikováno paralelní zpracování dat v rámci programu s cílem podstatného snížení nutné doby potřebné k jeho běhu. Byla identifikována problematická oblast a implementována dvě její různá řešení. První řešení paralelního zpracování dat bylo realizováno prostřednictvím asynchronního vzdáleného volání funkčního modulu, druhé řešení paralelního zpracování dat pak bylo implementováno pomocí procesů na pozadí.

Obě řešení byla úspěšně dokončena a každé z nich vedlo ke snížení nutné doby běhu programu určeného ke zpracování daného počtu materiálů jak oproti zcela původnímu programu, tak oproti programu již optimalizovanému.

Implementace s využitím asynchronního vzdáleného volaní funkčního modulu dosáhla nejlepších výsledků z hlediska nejkratší doby běhu programu a po otestování uživateli byla transportována do produkčního systému a v současné době je již nasazena v reálném provozu.

Hlavní cíl práce, analýza, návrh a implementace programu v programovacím jazyce ABAP pro ERP platformu SAP HANA a následné využití paralelního zpracování dat byl tedy splněn, program byl dokončen a v současné době je již nasazen v reálném provozu. V rámci práce byl navržen, implementován a optimalizován program splňující uživatelské požadavky a následně bylo implementováno paralelní zpracování dat dvěma odlišnými metodikami.

Další cíle této práce byly také splněny, ať už se jedná o popis teoretických pojmů a metodik souvisejících s vývojem a využití paralelního zpracování dat v programovacím jazyce ABAP pro platformu SAP HANA, nebo popis společnosti Siemens s. r. o., ve které byla práce realizována, včetně problému, který byl v práci řešen.

Využití paralelního zpracování dat se zcela přesvědčivě ukázalo jako správné rozhodnutí. Další zrychlení běhu programu je však již nyní omezeno zejména hardwarově, a to množstvím dostupných pracovních procesů na serveru.

V rámci řešení v diplomové práci bylo dosaženo několikanásobného zrychlení běhu zadaného programu. Je však nutné rovněž uvažovat o jeho následném nasazení a správě. Program vytvořený v rámci této práce bude s postupnými hardwarovými vylepšeními a zvýšením počtu dostupných pracovních procesů ještě zrychlovat svůj běh bez nutnosti jakéhokoli vnějšího zásahu vývojáře.

Paralelní zpracování dat představuje v aktuálním velmi konkurenčním výpočetním prostředí nutnost, jelikož násobné zrychlení běhu programu má kladný vliv na produktivitu všech uživatelů a zařízení, která jej využívají.

Seznam použité literatury

Odborná kniha

[1] BANDARI, Kiran. *Complete ABAP*. Boston, MA: Rheinwerk Publishing, 2016. 1047 p. ISBN 978-1-4932-1273-6.

[2] BARBARIC, Igor. *Design patterns in object-oriented ABAP*. 2nd ed. Boston, MA: Galileo Press, 2009. 254 p. ISBN 978-1-59229-263-9.

[3] BAUMGARTL, Axel and Devraj BARDHAN. *SAP S/4HANA: an introduction*. 2nd ed. Boston, MA: Rheinwerk Publishing, 2018. 511 p. ISBN 978-1-4932-1598-0.

[4] GAHM, Hermann et al. *ABAP development for SAP HANA*. 2nd ed. Boston, MA: Rheinwerk Publishing, 2016. 641 p. ISBN 978-1-4932-1305-4.

[5] GAHM, Hermann. *ABAP Performance Tuning*. Boston, MA: Rheinwerk Publishing, 2009. 348 p. ISBN 978-1-59229-289-9.

[6] HARDY, Paul. *ABAP to the future*. 3rd ed. Boston, MA: Rheinwerk Publishing, 2019. 864 p. ISBN 978-1-4932-1761-8.

[7] KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press, 2004. 278 s. ISBN 80-251-0342-0.

[8] KOSEOGLU, Kerem. *Design patterns in ABAP Objects*. Boston, MA: Rheinwerk Publishing, 2016. 388 p. ISBN 978-1-4932-1464-8.

[9] KÜHNHAUSER, Karl-Heinz. *ABAP: výukový kurz*. Brno: Computer Press, 2009. 365 s. ISBN 978-80-251-2117-7.

[10] MYSLÍN, Josef. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press, 2016. 256 s. ISBN 978-80-251-4650-7.

[11] ŠOCHOVÁ, Zuzana a Eduard KUNCE. *Agilní metody řízení projektů*. 2. vyd. Brno: Computer Press, 2019. 223 s. ISBN 978-80-251-4961-4.

[12] SOMMERVILLE, Ian. *Softwarové inženýrství*. Brno: Computer Press, 2013. 680 s. ISBN 978-80-251-3826-7.

[13] WOOD, James and Joe RUPERT. *Object-oriented programming with ABAP Objects*. 2nd ed. Boston, MA: Rheinwerk Publishing, 2016. 470 p. ISBN 978-1-59229-993-5.

[14] WOOD, James. *ABAP cookbook: programming recipes for everyday solutions*. Boston, MA: Galileo Press, 2010. 548 p. ISBN 978-1-59229-887-7.

Elektronické dokumenty a ostatní

[1] Agilealliance [online]. [cit. 2019-09-22]. Dostupné z: https://www.agilealliance.org/glossary/backlog/

[2] Agileconnection, [online]. [cit. 2019-09-20]. Dostupné z: https://www.agileconnection.com/article/kickoff-meetings-give-your-agile-projects-running-start

[3] Businessnsoftware, [online]. [cit. 2019-11-08]. Dostupné z: http://www.businessnsoftware.com/erp-market-share/

[4] Cleverism, [online]. [cit. 2019-12-03]. Dostupné z: https://www.cleverism.com/skills-and-tools/abap/

[5] Erp-information, [online]. [cit. 2019-08-02]. Dostupné z: https://www.erp-information.com/history-of-erp.html

[6] Guru99, [online]. [cit. 2020-01-07]. Dostupné z: https://www.guru99.com/dbms-transaction-management.html

[7] Medium, [online]. [cit. 2020-01-14]. Dostupné z: https://medium.com/cr8resume/make-you-hand-dirty-with-mvp-model-view-presenter-eab5b5c16e42

[8] Mulesoft, [online]. [cit. 2020-02-26]. Dostupné z: https://www.mulesoft.com/resources/api/sap-hana-system-migration

[9] Oracle Corp., [online]. [cit. 2020-01-16]. Dostupné z: https://docs.oracle.com/cd/A97330_01/integrate.902/a95438/rfc.htm

[10] Oracle Corp., [online]. [cit. 2019-08-12]. Dostupné z: https://www.oracle.com/applications/erp/what-is-erp.html

[11] SAP, Ltd., [online]. [cit. 2020-03-04]. Dostupné z: https://help.sap.com/doc/saphelp_nw70/7.0.31/en-US/fa/096e92543b11d1898e0000e8322d00/content.htm

[12] SAP, Ltd. [online]. [cit. 2020-04-05]. Dostupné z: https://news.sap.com/2020/03/ten-years-sap-hana-era/

[13] Scrum, [online]. [cit. 2019-10-17]. Dostupné z: https://www.scrum.org/resources/scrum-guide

[14] Scrum, [online]. [cit. 2019-10-12]. Dostupné z: https://www.scrum.org/resources/what-is-a-product-owner

[15] Scrum, [online]. [cit. 2019-09-08]. Dostupné z: https://www.scrum.org/resources/what-is-a-scrum-development-team

[16] Scrum, [online]. [cit. 2019-09-08]. Dostupné z: https://www.scrum.org/resources/what-is-a-scrum-master

[17] Scrum, [online]. [cit. 2019-09-05]. Dostupné z: https://www.scrum.org/resources/what-is-scrum

[18] Siemens, s. r. o., [online]. [cit. 2019-07-01]. Dostupné z: https://new.siemens.com/cz/cs/spolecnost/o-nas/historie.html

[19] Siemens, s. r. o., [online]. [cit. 2019-07-20]. Dostupné z: https://new.siemens.com/global/en/company/about/history/company/1847-1865.html

[20] Solutiondots, [online]. [cit. 2019-09-28]. Dostupné z: https://solutiondots.com/blog/sap-erp-modules/

[21] Theserverside, [online]. [cit. 2019-11-22]. Dostupné z: https://www.theserverside.com/definition/J2EE-Java-2-Platform-Enterprise-Edition

[22] Tutorialspoint, [online]. [cit. 2019-08-27]. Dostupné z: https://www.tutorialspoint.com/sap/sap_introduction.htm

[23] Versaccounts, [online]. [cit. 2019-08-14]. Dostupné z: http://www.versaccounts.com/blog/the-history-of-erp-systems/

[24] Visual-paradigm [online]. [cit. 2019-11-08]. Dostupné z: https://www.visual-paradigm.com/scrum/what-is-sprint-backlog-in-scrum/

[25] Yodiz, [online]. [cit. 2020-01-06]. Dostupné z: https://www.yodiz.com/blog/best-practices-for-sprint-planning-meeting-in-agile-project-management/

Seznam zkratek

ABAP	Advanced Business Application Programming
Corp	Corporation
ČR	Česká republika
ECC	Enterprise Resource Planning Central Component
ERP	Enterprise Resource Planning
HANA	High-performance Analytic Appliance
J2EE	Java 2 Enterprise Edition
Ltd	Limited
MM	Material Management
MRP	Material Resource Planning
MVC	Model-View-Controller
MVP	Model-View-Presenter
RFC	Remote Function Call
S/4 HANA	SAP Business Suite for SAP HANA
SAP	Systems, Applications and Products
SAP R	SAP Real-Time Processing
s. r. o.	společnost s ručením omezeným

Prohlašuji, že

- jsem byl(a) seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst. 3);
- souhlasím s tím, že diplomová práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO. Souhlasím s tím, že bibliografické údaje o diplomové práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, diplomovou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 24.04.2020

Hošna

.....

Bc. Jan Hošna

Seznam příloh